FUTURA

GETTING STARTED ON YOUR ORIC

Tim Hartnell and Peter Shaw

GETTING STARTED ON YOUR ORIC

Also in this series

GETTING STARTED ON YOUR VIC 20
GETTING STARTED ON YOUR ATARI
GETTING STARTED ON YOUR BBC MICRO
GETTING STARTED ON YOUR ZX81
GETTING STARTED ON YOUR SPECTRUM
GETTING STARTED ON YOUR DRAGON

Tim Hartnell & Peter Shaw

Getting Started on Your Oric

Futura
Macdonald & Co
London & Sydney

A Futura Book

First published in Great Britain in 1983
by Futura Publications
A Division of Macdonald & Co. (Publishers) Ltd
London & Sydney

ISBN 0 7088 2447 1

Reproduced, printed and bound in Great Britain by
Hazell Watson & Viney Ltd,
Member of the BPCC Group,
Aylesbury, Bucks

Futura Publications
A Division of
Macdonald & Co (Publishers) Ltd
Maxwell House
74 Worship Street
London EC2A 2EN

A BPCC plc Company

To
Keith, Neil, Julian & Roy

A NOTE ABOUT THE AUTHORS:

Tim Hartnell is the most widely published author of books related to personal computers in the world. He is a leading computer expert and journalist, who has contributed extensively to the technical computer press. His books include *Getting Acquainted With Your ZX81*, *Let your BBC micro Teach you to Program* and *The Personal Computer Guide*.

Peter Shaw is a sixteen year old programmer from Stanwell, in Middlesex, who has worked extensively for computer magazines such as Interface, ZX Computing and Home Computing Weekly. He has also written books like *Games for your ZX Spectrum* and *Games for your Oric*.

ACKNOWLEDGEMENTS

I would like to thank Tim, Liz and Richard for their support with this book, Clive and Sue for their help at the computer club, Ginette for deciphering my scrawling on Tim's manuscript and all the other people in the computer industry who have taken an interest in my work.

CONTENTS

Chapter		
1	DISCOVERING THE KEYBOARD	1
2	PRINTING THINGS ON THE SCREEN	4
3	RINGING THE CHANGES	14
4	DESCENT INTO CHAOS	19
5	ROUND AND ROUND WE GO	25
6	CHANGING IN MID-STREAM	37
7	SOUNDING OUT	44
8	MAKING COMPARISONS	52
9	A GAME AND A SPEED TEST	56
10	STRINGING ALONG	63
11	READING DATA	73
12	LIVING MORE COLOURFULLY	84
13	GETTING LISTED	100
14	LET THE GOOD GRAPHICS ROLL	108
15	GETTING TAPED	119
16	MORE GREAT GAMES	122
17	USING THE SYSTEM	149
APPENDIX: USING MATHS		151

INTRODUCTION

Welcome to *Getting Started on your Oric*. You have in your hands a guide to one of the most advanced pieces of equipment you can buy today — a microcomputer.

This book will take you, step by step, through BASIC programming, showing you many features of the Oric 1. We will explain its colour, sound and high resolution graphics in detail so that you will be able to include all three in your own programs comfortably. And you will be able to write programs, we assure you.

Learning BASIC on the Oric is going to be painless, in fact, with this book, it's going to be *fun*. The Oric uses a version of BASIC called Microsoft, a version widely accepted as standard on most microcomputers, so if you decide to buy another computer after you've learnt to use the Oric then you will not have too much trouble making the change.

Come on and join us now, as together we discover how we can make the most of your Oric 1.

Tim Hartnell
Peter Shaw
London, August 1983

CHAPTER ONE
DISCOVERING THE KEYBOARD

You've just bought a great computer, and in this book we're going to show you how to get the most out of it. Don't worry if this is the first computer you have ever owned. We're going to take things slowly, and in small steps, so you'll have no trouble keeping up with us.

READ THE BOOK WITH YOUR COMPUTER TURNED ON

It is vital that you have the computer turned on at all times when you read this book, (or at least the first time you read through it). This is an *action book*, and unlike a novel, it is not designed simply to be read. This book is like a book on how to drive, you can never learn about changing gears, and how it feels to handle a car in traffic, without actually taking command of a vehicle. So it is with your computer and this book. Try out each new command and function when it is explained to you, experiment with the games and other programs, and you'll find you're learning to program without even trying.

THE KEYBOARD

First we have to find our way around the keyboard. It looks like a small typewriter keyboard with a few extra keys, and this is what it is. It is surprising how quickly it will become familiar to

you, in fact, by the time you have worked through this book you will probably be able to type with some competence.

THE BASIC KEYBOARD

Leaving out the extra keys for a moment, let's look at the main part of the keyboard.

This part is known as a QWERTY keyboard. If you look at the second row down you'll see the first six letters spell QWERTY. Try typing something in, like your name. As you type, the letters will appear on the screen and you will also hear a small beep every time you press a key, this is so that you can type something in and, without looking at the screen, you know it has been entered. Practice as much as you can with the keyboard, sentences like *The quick brown fox jumped over the lazy dogs* are ideal, because they use every letter of the alphabet. Once you are happy with the QWERTY keyboard we can then move on to the extra keys.

DELETE

By now the screen may be looking pretty crowded, and something like a disaster area. If you look at the right hand end of the second row you'll see a key marked *DEL*, this is short for DELETE. If you hold down this key it will move back over what you typed, erasing as it goes.

Another way of clearing up 'rubbish' like this is to turn off the power for a short while, but this is not advised as it causes the contents of the memory to be erased.

THE SHIFT KEYS

At both ends of the fourth row of keys down you'll find keys marked SHIFT. Their effect is much the same as it is on an ordinary typewriter, when you're in lower case mode, (we'll talk about that later), if you press shift and a letter key then you get the letter in upper case. If you press shift and a numeric key

then you get the symbol above the key, i.e. press SHIFT/2 you will see the @ symbol appear on the screen.

THE CURSOR KEYS

On either side of the space bar you'll see four curious looking keys with arrows above them. These keys are called *cursor* keys. A cursor is that flashing blob which has traced your typing on the screen, this is there to tell you where the next character you type will appear on the screen. Anyway, to get back to the point of those keys, try pressing the key with the *down* arrow above it. Did you expect that to happen? Try pressing the other cursor keys, you should know what will happen now.

CHAPTER TWO
PRINTING THINGS ON THE SCREEN

THE RETURN KEY

The key that you'll probably use most often is RETURN. Whenever you're typing information into the computer, it will wait until you press RETURN before acting on the material you've entered. If you've written a program line, (more of that later), it will ignore that line even if you have pressed RETURN, i.e. if you type in PRINT 2 and then RETURN the computer will display:

```
PRINT 2
2
```

But if you type 1Ø PRINT 2 and then press RETURN the computer will display:

By adding a number in front of the command you have made it a *program line*. The computer will not act upon the command until you type RUN, type RUN now, do you see what has happened? The computer has RUN your first program, we will now explain what your program is.

THE PRINT COMMAND

Type LIST and the RETURN, you should see your program:

```
10 PRINT 2
```

As we explained before, the number 10 is a line number, if you type in

```
5 PRINT 1
```

then type LIST, (LIST is a command to list your program), you should see:

```
5 PRINT 1
10 PRINT 2
```

Isn't that good, the computer puts line 5 in front of line 10, clever huh? Now look at that word following the line number, PRINT. If you RUN the program then you should see how PRINT works. It takes the information which follows the PRINT command, with a few exceptions which we'll learn about in a moment, and PRINTs this on the screen, which after all is exactly what you'd expect it to do.

But your computer is cleverer than that. If the word PRINT is followed by a sum, it will work out the answer before it PRINTs, and it will PRINT the answer. Try it now. Enter the following line:

```
PRINT 5+3
```

You should see the figure 8 appear. The computer added 5 and 3 together, as instructed by the plus (+) sign, then printed the result on the screen. It can do subtraction as well. Type in this, and remember to type RETURN afterwards.

```
PRINT 8-3
```

Now the computer can — of course — do a wide range of mathematical tasks, many of them far more sophisticated than simple addition and subtraction. But, there is a slight hitch. When it comes to multiplication, the computer does not use the × sign which you probably used at school. Instead, it uses an asterisk (*), and for division, instead of ÷ the computer uses a slash sign (/).

DOING MORE THAN ONE THING AT ONCE

The computer is not limited to a single operation in a PRINT statement. You can combine as many as you like. Try the next one, which combines a multiplication and a division. Type it in, then press RETURN to see the computer evaluate it:

```
PRINT 7*8/5
```

This seems pretty simple. Just type in PRINT followed by the material you want it to PRINT, and that's all there is to it. But, it's not as simple as that! Try the next one out and see what happens:

```
PRINT TESTING
```

That doesn't look too good. Instead of the word TESTING we've got the number Ø. To explain briefly what has happened, let's consider the pocket calculator. On your calculator you can possibly store away numbers by pressing the Memory Plus button. On your Oric you can also store away a number, in fact you can store away quite a few numbers. When you told the computer to PRINT TESTING you were asking the computer to print one of the numbers that it had stored away in its memory. But, as you hadn't told the computer what TESTING actually equals then it takes the value to be Ø. Foolish machine. Computers may be very, very clever machines, but they need to be led by the hand, like a very stupid child who has to be told exactly what to do. Give them the right instruction and they will do the right thing. But give them the wrong instruction or — even worse — confuse them, and they will give up in despair, or do something quite alien to your intentions.

STRINGS

If you want the computer to print the word TESTING you must put quote, or speech marks around the word, like this:

```
PRINT "TESTING"
```

This time when you press RETURN the word TESTING will appear on the screen. This is worth remembering. When you

want the computer to print out a word, or a whole lot of words, symbols, spaces and numbers, you need to put quote marks around the material you want to print. Information held in this way, between quote marks, is called a most peculiar name in computer circles. The jargon for information enclosed in quote marks is a *string*. So, in our example above, the word TESTING, when enclosed by quote marks, is called a string.

OUR SECOND PROGRAM

Type the following into the computer. Notice each line starts with a number, as explained before, this is a *line number*. Remember to press RETURN at the end of each line.

```
10 PRINT "JACK AND JILL"
20 PRINT "WENT UP THE HILL"
30 PRINT "TO FETCH A PAIL"
40 PRINT "OF WATER"
```

When you have typed the whole program into the computer, type LIST, press RETURN and check that the program on the screen is the same as the one listed above. RUN the program as you did before, did the computer do what you expected it to do? If it did something drastically different, i.e. gave you an error message, then check that you typed everything in correctly, and RUN the program again.

GETTING THINGS INTO ORDER

Look at the program again, do you notice how line 20 follows line 10? And line 30 is before 40? Of course you would expect no less from your computer, after all, it can *count*! But what if we typed in another line, say line 25. The computer would rearrange the list of instructions so that line 25 came between line 20 and line 30. This is useful if you forgot an instruction, or you want to add a line, it is also why most programs are numbered in steps of 10.

Type in the following line.

```
25 REM A LINE IN THE MIDDLE
```

Now, LIST the program, it should look like this:

```
10 PRINT "JACK AND JILL"
20 PRINT "WENT UP THE HILL"
25 REM A LINE IN THE MIDDLE
30 PRINT "TO FETCH A PAIL"
40 PRINT "OF WATER"
```

As you can see, the line numbered 25 has moved itself into the correct position in the program, between lines 20 and 30. Now RUN the program.

MAKING REMARKS

You should find that the new line, line 25, has not made any difference at all to the running of the program. Why not? Why did the computer decide to ignore line 25? The word REM stands for *remark* and is used within programs when we want to include information for human use only. You'll find REM statements scattered throughout the programs in this book. In each and every case the computer ignores the REM statements. They are only there for your convenience, for the convenience of the programmer, or for someone else reading the program.

Often you'll use REM statements at the beginning of the program, like this one:

```
5 REM JACK AND JILL POEM
```

You may wonder why this would be necessary. After all, it is pretty obvious that the computer is holding the *Jack and Jill* poem, even without the line 5 REM statement. You are right.

In this case there is little point in adding a title REM statement to this program. But have a look at some of the programs further on in this book. Without REM statements you'd have a pretty difficult time trying to work out what the program was supposed to do.

REM statements are often scattered throughout programs. There they serve to remind the programmer what each section is supposed to do. Once you've been programming a while, you'll be amazed at how many programs you'll collect in listing form which — when you go back to them in a month or so — will be totally obscure. You won't have a clue how the program works or, even more important, what on earth it is, or what it is supposed to do. This is where you will find REM statements invaluable.

It is worth getting into good habits early as a programmer. So, we suggest you start right now adding REM statements to your programs. If you come across programs, or program fragments in this book, which you want to keep, and which do not have REM statements, get into the habit of using REM statements by adding them to these programs. And make sure you use them in your original programs.

BACK TO PRINT

Let's return to the PRINT command. Empty your computer's memory by typing NEW and then PRESS return, you have lost the contents of the computer's memory, NEW is a pretty drastic command, and should only be used when you are sure that you've finished with a program. You're ready for another program, but first let's learn a little trick which keeps things tidy. Press the CTRL and at the same time press the L key. The screen should have suddenly cleared, if it did not then send your Oric back to Tangerine for immediate replacement!

MORE PRINT STATEMENTS

Type in the following program and RUN it:

```
10 PRINT 1,2
15 PRINT
20 PRINT 1;2;3
25 PRINT
30 PRINT "ORIC"
```

```
35 PRINT
40 PRINT "23+34=";23+34
45 PRINT
50 PRINT "2*3=";2*3
55 PRINT
60 PRINT 3^5
65 PRINT
70 PRINT "THE ANSWER IS ";23+5-7/6
```

You should get something like this on the screen:

```
1  2

1 2 3

ORIC

23+34=57

2*3=6

243

THE ANSWER IS 26.8333334
```

Now there's a lot we can learn from this program.

Firstly, as in the *Jack and Jill* program, the computer executes a program line by line, starting at the lowest number and proceeding through the line numbers in order until it runs out of numbers, when it stops. (You'll discover that this orderly progression of line numbers does not always apply, as there are ways of making the computer execute part of a program out of

strict numerical order, but for the time being it is best to assume that a program will be executed in order).

Look at the first line of your program, line 1Ø. You can see that there is a comma between the 1 and the 2. This has the effect of getting the computer to print the first number (1) on the left hand side of the screen, and the second number (2) five spaces further on. When you use a comma like this to divide things which follow a PRINT statement (but not when the comma is in a *string*, that is between quote marks), it divides the screen into parts, each part five spaces long, printing the information which follows the comma in the next available part.

The second line of the program, line 15, is just the word PRINT with nothing following it. This has the effect, as you can see in the printout (and on the television screen) of putting a blank line between those lines which do include information after them. (The same comment applies to lines 25, 35, 45, 55 and 65).

Line 2Ø has three numbers (1, 2 and 3) separated not by commas (as in line 1Ø) but by semicolons. This has the effect of putting a single space between the numbers. This single space only applies to numeric information, when you come to strings then there are no spaces put in between the information. We suggest you experiment by using the semicolon.
Line 3Ø is the word *ORIC* and this is a ..... If you mentally said *string* when you came to those dots, then you're learning well. The word is a string, in computer terms, because it is enclosed by quote marks.

Line 4Ø is rather interesting. For the first time we have included numbers and a symbol (=) within a string. As you can see, the computer prints *exactly* what is within the quote marks, but works out the result of the calculation for the material outside the quote marks, giving — in this case — the result of adding 23 to 34. Try to remember that the computer considers everything within quote marks as words, even if it is made up from

numbers, symbols, or even just spaces, or any combination of them, while it counts everything that is *not* within quote marks in a PRINT statement as a number.

So line 40 treats the first part, within quote marks, as a string, and the second part, outside quote marks, as numerical information which it processed.

In line 50 we see the asterisk (*) used to represent multiplication and the computer quite reasonably works out what 2 times 3 is and prints the answer 6. In line 60 we come across a new, and strange sign, a little upward arrow. This means 'raise to the power of' so line 60 means PRINT 3^5. Now, it is pretty difficult for a computer to print another number halfway up the mast of another number, so we use the upward arrow to remind you (by pointing upward) that it really means 'print the second number up in the air'. You probably know that 7^2 is read as 'seven squared'. It means raise seven to the second power. In a similar way, 3^5 means raise three to the fifth power which is exactly what the computer does in line 60.

The final line of this program combines a string ('the answer is') with a numerical information ($23+5-7/6$). You can see that, as expected, the computer works out the sum before printing the answer, and prints the string exactly as it is. Look closely at the end of the string. You'll see there is a space there.

After the closing quote is a semicolon (;) which, as we learned in line 20, as far as strings are concerned, joins various elements of a PRINT statement together. This semicolon means that the result of the calculation is printed up next to the end of the string. However, if there were no space within the quote marks, the result of printing line 70 would be:

```
THE ANSWER IS26.8333334
```

As you can see, this is by no means as clear as the original version.

That brings us to the end of the second chapter of the book. We're sure you're pleased at how much you've learned so far, and are looking forward to continuing your learning. But now you've earned a break. So take that break, and then come back to the book to tackle the third chapter.

CHAPTER THREE
RINGING THE CHANGES

It's all very well getting things onto the computer's screen, as we learnt to do in the last chapter, but from time to time you'll discover we need to be able to get printed material *off* the screen during a program. We can't use that little trick we learnt before because the computer can't reach out of the case and press the necessary keys. What we need is a command that will clear the screen. We can do this with the command CLS, which stands for Clear the Screen.

CLEAR THE SCREEN

Enter the following program into your computer and run it.

```
10 PRINT "TESTING"
20 INPUT A$
30 CLS
```

When you run the program, you'll see the word TESTING appear on the screen, more or less as you'd expect:

```
TESTING
?
```

However, can you see the flashing cursor below the word TESTING and behind a question mark.

This is an *input prompt*. An input prompt, which appears in a program when the computer hits the word INPUT, means that the computer is waiting for you to enter something from the keyboard. You'll recall that we spoke earlier about *strings* and about how they were anything which was enclosed by quotes. The computer signals that it is talking about a string by using the dollar sign. So line 2Ø is waiting for a string called A$ to be INPUT by you.

So what are we going to do about that flashing cursor? You will have to type something in, your name will do, it doesn't make much difference to the way the program runs.

When you have finished typing your message, remember to press RETURN, or the computer won't know that you've finished.

Once you've done this the computer moved along to line 3Ø where it found the instruction CLS. As you know this means Clear the Screen, and what did your program do? yes it cleared the screen.

Run the program a few more times, until you've got a good idea of what is happening, and you've followed through — in your mind — the sequence of steps the computer is executing.

DOING IT AUTOMATICALLY

Instead of waiting for you to enter something from the keyboard, you can write a program which clears the screen automatically, as in our next example. Enter this next program into your computer, type RUN and press RETURN, and sit back for the amazing Flashing Word demonstration.

```
10 PRINT "AUTOTESTING"
20 FOR A=1 TO 100
30 NEXT A
40 CLS
50 FOR A=1 TO 100
60 NEXT A
70 RUN
```

In this program you'll see the word *AUTOTESTING* alternately flashing off and on at the top of the screen. What's happening here? Let's look at the program, and go through it line by line.

Firstly, as you know, line 10 prints the word *AUTOTESTING* on the screen. Next, the computer comes to line 20, where it meets the word FOR. We'll be learning about FOR/NEXT loops (as they are called) in detail in a later chapter, but all you need to know here is that the computer uses them for *counting*. In this program line 20 and 30 (FOR is in line 20 and NEXT in line 30) tells the computer to count from one to one hundred, before moving on. As you can see, it does this counting pretty quickly.

So, it waits for a moment while counting from one to a hundred. Then it comes to line 40, which is the command CLS, which tells the computer to clear the screen. It does this, and *AUTOTESTING* disappears from the screen. The computer then encounters, in lines 50 and 60, another FOR/NEXT loop, so it waits a while as it counts from one to one hundred again. Continuing on in sequence, it comes to line 70, where it finds the word RUN. Now, as you know, you get the computer to execute a program by typing in RUN, followed by RETURN. But RUN is a command, so there is no reason why you cannot use it *within* a program, as we have in line 70 of this program. When the computer comes to line 70, it obeys the command,

and RUNs the program again, so that it goes through the program again and again.

CHANGING PROGRAM LINES EASILY

Your computer is provided with an EDIT function, which makes it very simple to change the content of the lines within a program. Pull the plug out on the last program, then enter the following program into your computer. *Do not RUN the program*. We want to explain something about the program before you do:

```
10 REM AN EDIT TEST
20 PRINT "TESTING"
30 PRINT "EDIT TESTING"
```

Enter as a direct command EDIT 10. You should see line 10 appear on the screen, just below your last command. Now, using the *cursor keys*, which as we said before, are on either side of the space bar, move the cursor so that it is flashing just before the number 10, like this:

```
▒10 REM AN EDIT TEST
```

You will now have to enter the part of this line you want to keep into the computer. To do this, put one finger on the CTRL key, and another finger on the A key. The cursor should move over one position. Keep pressing this key until the cursor is flashing over the T in TEST. Then type over the rest of the line changing TEST for CHANGE. When you have typed in CHANGE, press RETURN and LIST the program. It should now look like this:

```
10 REM AN EDIT CHANGE
20 PRINT "TESTING"
30 PRINT "EDIT TESTING"
```

Before you read on, take some time changing lines 20 and 30 as well as line 10. But remember that you will have to put speech marks around anything you change in the last two lines, or you will be told that you have made a mistake.

GETTING THE PROGRAM BACK

We have already covered the word LIST, but there are a number of useful things that you can do with it. When you tell the computer to LIST the program, you needn't LIST all of it, you can list part of it, or just a single line. To list just one line type LIST, followed by the line you want to examine. If you want to list just part of your program then type LIST, followed by the first line number, a minus sign, and then the last line you want to see, i.e.:

```
LIST 20-30
```

You can also omit the first or last number, i.e. if you want to list the program to line 20 then type:

```
LIST -20
```

If you want to list the program from line 20 onwards then type:

```
LIST 20-
```

USING A PRINTER

There are two commands which trigger the printer, and they are very simple to use and remember. They are LPRINT, which stands for Line Print, and LLIST, which stands for Line List. If you are thinking that you have seen commands similar to those before somewhere, then you'd be right. The first one LPRINT, has the same effect as PRINT, but only sending the information which follows to the printer, instead of the screen. The second command — LLIST — LISTs the whole program on the printer, so that you have got a permanent copy of your program.

CHAPTER FOUR
DESCENT INTO CHAOS

It's time now to start developing some real programs. You'll notice that from this point in the book there are some rather lengthy programs. Many of them will contain words from the BASIC programming language which have *not* been explained. This is because, as programs become more complex (and far more satisfying to run) it becomes more and more difficult to keep programming words which have not been explained out of the program. However, it is not a major problem.

We are working methodically through the commands available on the computer and, in due course, all of them will be covered. When you come across a word in a program which seems unfamiliar, just enter it into the computer. You'll find that you'll soon start picking up the meaning of words which have not yet been explained, as you see how they are used within a program. So, if you find a new word, don't worry. The program will work perfectly without you knowing what the word is, and investigating the listing after you've seen the program running is likely to allow you to work out what it means anyway.

RANDOM EVENTS

In the world of nature, as opposed to the manufactured world of man, randomness appears to be at the heart of many events.

The number of birds visible in the sky at any one time, the fact that it rained yesterday and may rain again today, the number of trees growing on the side of a mountain, all appear to be somewhat random. Of course, we can predict with some degree of certainty whether or not it will rain, but the success of our predictions appears to be somewhat random as well. When you toss a coin in the air, whether it lands heads or tails depends on chance. The same holds true when you throw a six-sided die in the air. Whether it lands with the one, the three or the six showing depends on random factors.

Your computer has the ability to generate random numbers which are very useful for getting the computer to imitate events in the real world. On the Oric you must use a function which looks like this:

RND (*n*)

You will notice that the *n* must be replaced by an integer, (whole number). If the number is greater or equal to 1, then the number generated is between Ø and 1. If *n* equals Ø, then the most recently generated number is produced. If *n* is less than zero, then the same number is produced for each *n*. The one that we're interested in is the first one, to produce a random number between Ø and 1. To see RND in action, try the following program:

```
10 PRINT RND(1)
20 GOTO 10
```

When you run it, you'll see a list of numbers like this appear on the screen:

```
 .623695763
 .519966953
 .099633678
 .212823365
 .161374366
 .203185255
```

```
.677045022
.17392633
.588175519
.754821112
.429002078
```

If you leave it running, it will go on and on apparently forever, writing up new random numbers on the screen.

Now random numbers between zero and one are of limited interest if we want to generate the numbers and get them to stand for something else. For example, if we could generate 1's and 2's randomly, we could call the 1's *heads* and the 2's *tails* and use the computer as some kind of electronic 'coin'. If we could get it to produce whole numbers between one and six, we could use the computer as a six-sided die.

Fortunately, there is a way to do this. Enter the next program and run it. Please note that there is a space between the quotes in line 10.

```
10 PRINT INT (RND(1)*6)+1;" ";
20 GOTO 10
```

When you run this program, you'll get a series of numbers (chosen at random between 1 and 6) like these:

```
1  2  5  5  3  1  3  4  4  6  1  5  1  1
  1  1  1  1  1  3  1  4  2  2  6  5  4
 3  1  2  6  5  5  4  2  2  2  3  6  6
2  2  5  6  6  6  5  4  5  2  3  5  5  1
```

Even though we can create vast series of numbers between 1 and 6 with a program like this to our heart's content, it is not particularly interesting. And, if you ran the program over and over again, you might find that the sequence of numbers may

start to become a little familiar. The random numbers may stop looking so random.

This is because the computer does not really generate random numbers, but only looks as if it is doing so. Inside its electronic head, the computer holds a long, long list of numbers, which it prints in order when asked for random numbers. The list is so long, it is almost impossible to see any pattern or repetition in it. However, this is fine for our purposes.

LOWER THE TONE

Up until now we have been working totally in upper case. So that you don't think that's all the Oric can do, put your finger on the CTRL key and at the same time press the T key. If you type in some letters now, you'll find they will all be lower case. Unfortunately the Oric can't think in lower case, try typing in something like list. You should get something on the screen which looks like this:

```
? SYNTAX ERROR
```

Not at all what you'd expect, you did type in a 'legal' command, didn't you? The computer needs all commands to be entered in upper case, you can use lower case inside a string, but all commands *have* to be entered in upper case.

FAST FOOD CRAZINESS

Enter and run the next program, which makes an interesting use of the computer's ability to generate random numbers. As you can see, it creates a scene where you have turned up at a fast food outlet, desperate for something to eat, and you've decided to let random number generator pick your food for you:

```
10 REM FAST FOOD
30 A=INT (RND(1)*4)+1
40 PRINT "You've ordered..."
50 IF A=1 THEN PRINT"A hamburger"
60 IF A=2 THEN PRINT"French Fries"
70 IF A=3 THEN PRINT"Spare Ribs"
80 IF A=4 THEN PRINT"Hot Dogs with ketch
up"
90 PRINT
100 GOTO 30
```

When you run this, you'll get something like this list of food on the screen:

```
You've ordered...
Spare Ribs

You've ordered...
Hot Dogs with ketchup

You've ordered...
French Fries

You've ordered...
Spare Ribs
```

Notice how the program sets the letter A to the value of the random number in line 3Ø. In this case, the letter A is *standing for* a number. It is called a *variable*, or, because it stands for a number (as opposed to standing for a word, or a string) it is called a *numeric variable*. In computer jargon, we say that, (in line 3Ø), the computer has *assigned* the value of the random number to the variable A. And, as you can see in lines 5Ø, 6Ø, 7Ø and 8Ø, the value assigned to A determines which food order you place. Read this over if it seems incomprehensible the first time.

CHAPTER FIVE
ROUND
AND ROUND WE GO

In this chapter we'll be introducing a very useful part of your programming vocabulary — FOR/NEXT loops. You'll recall that we mentioned FOR/NEXT loops when demonstrating the use of CLS to clear the screen. There, a loop was used to add a delay after the word was printed on the screen before the screen was cleared.

A FOR/NEXT loop is pretty simple. It has the form of two lines in the program, the first like this:

```
FOR A=1 TO 20
```

And the second as follows:

```
NEXT A
```

The *control variable*, the letter after FOR and after NEXT must be the same.

As a FOR/NEXT loop runs, the computer counts from the first number up to the second, as these two examples will show:

```
10 FOR A=1 TO 20
20 PRINT A;
30 NEXT A
```

When you run it, this appears on the screen:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
17 18 19 20
```

Here's another version:

```
10 FOR A=765 TO 781
20 PRINT A;
30 NEXT A
```

And this is the result of running it:

```
765 766 767 768 769 770 771 772 773 774
775 776 777 778 779 780
781
```

STEPPING OUT

In the two previous examples, the computer has counted up in one, but there is no reason why it should do so. The word STEP, can be used *after* the FOR part of the first line as follows:

```
10 FOR A=10 TO 100 STEP 10
20 PRINT A;
30 NEXT A
```

When you run this program, you'll discover it counts (probably as you expected) in steps of 10, producing this result:

```
10 20 30 40 50 60 70 80 90 100
```

STEPPING DOWN

The STEP does not have to be positive. Your computer is just as happy counting backwards, using a negative step size:

```
10 FOR A=100 TO 0 STEP -15
20 PRINT A;
30 NEXT A
```

This is what the program output looks like:

```
100  85  70  55  40  25  10
```

MAKING A NEST

It is possible to place one or more FOR/NEXT loops *within* each other. This is called *nesting loops*. In the next example, the B loop is *nested* inside the A loop:

```
10 FOR A=1 TO 3
20 FOR B=1 TO 2
30 PRINT A;"TIMES ";B;"IS ";A*B
40 NEXT B
50 NEXT A
```

The nested programs produce this result:

```
 1 TIMES 1 IS 1
 1 TIMES 2 IS 2
 2 TIMES 1 IS 2
 2 TIMES 2 IS 4
 3 TIMES 1 IS 3
 3 TIMES 2 IS 6
```

You must be very careful to insure that the *first* loop started is the *last* loop finished. That is, if FOR A . . . was the first loop you mentioned in the program, the last NEXT must be NEXT A.

Here's what can happen if you get them out of order (the B loop ends *after*, rather than before, the A loop):

```
10 FOR A=1 TO 3
20 FOR B=1 TO 2
30 PRINT A;"TIMES ";B;"IS ";A*B
40 NEXT A
50 NEXT B

1 TIMES 1 IS 1
2 TIMES 1 IS 2
3 TIMES 1 IS 3
?NEXT WITHOUT FOR ERROR IN 50
```

MULTIPLICATION TABLES

You can use nested loops to get the computer to print out the multiplication tables, from one times one right up to twelve times twelve, like this:

```
10 FOR A=1 TO 12
20 FOR B=1 TO 12
30 PRINT A;"TIMES ";B;"IS ";A*B
40 NEXT B
50 NEXT A
```

Here's part of the output:

```
4 TIMES 3 IS 12
4 TIMES 4 IS 16
4 TIMES 5 IS 20
4 TIMES 6 IS 24
4 TIMES 7 IS 28
4 TIMES 8 IS 32
4 TIMES 9 IS 36
```

```
4 TIMES 10 IS 40
4 TIMES 11 IS 44
4 TIMES 12 IS 48
```

There is no reason why both loops should be travelling in the same direction (that is, why both should be counting upwards), as this variation on the Times Table program demonstrates:

```
10 FOR A=1 TO 12
20 FOR B=12 TO 1 STEP -1
30 PRINT A;"TIMES ";B;"IS ";A*B
40 NEXT B
50 NEXT A
```

Here's part of the output of that program:

```
2 TIMES 12 IS 24
2 TIMES 11 IS 22
2 TIMES 10 IS 20
2 TIMES 9 IS 18
2 TIMES 8 IS 16
2 TIMES 7 IS 14
2 TIMES 6 IS 12
2 TIMES 5 IS 10
2 TIMES 4 IS 8
2 TIMES 3 IS 6
```

CRACKING THE CODE

It's time for our first *real* program. In this program which uses several FOR/NEXT loops, CODEBREAKER, the computer thinks of a four-digit number (like 5462) and you have eight guesses in which to work out what the code is. In this program, written by Adam Bennett and Tim Summers, you not only have

to work out the four numbers the computer has chosen, but also determine the order they are in.

After each guess, the computer will tell you how near you are to the final solution. A 'white' is the right digit in the wrong place in the code, a 'black' is the right number in the right place. You'll see that you are aiming to get four blacks. When you have, you have cracked the code. Digits may be repeated within the four-number code. Enter the program, and play a few rounds against the computer. Then, return to the book for a discussion on it, which will highlight the role played by the FOR/NEXT loops.

```
5 CLS
10 PRINT"*******************************
*******"
20 PRINT
30 PRINT"ODEBREAKER BY A.J.B, T.S. AND P
.J.S."
40 PRINT:PRINT
50 PRINT"*******************************
*******"
70 PRINT"When you are told to do so, ent
er a"
80 PRINT"four-digit number and then prss
 "
90 PRINT"RETURN."
100 PRINT"Digits can be repeated."
110 PRINT"You have 8 goes to break the c
ode!"
120 PRINT"-----------------------------
--------"
140 WAIT 350
150 CLS
```

```
160 DIM B(4)
170 DIM D(4)
180 H=0
190 FOR A=1 TO 4
200 B(A)=INT (RND(1)*10)
210 NEXT A
220 FOR C=1 TO 8
230 PRINT "Enter guess number ";C;
240 INPUT X:IF X>9999 THEN 240
250 PRINT
260 IF INT(X/1000)=0 THEN PRINT "0";
270 PRINT X;
280 P=INT (X/1000)
290 Q=INT ((X-1000*P)/100)
300 R=INT ((X-1000*P-100*Q)/10)
310 S=(X-1000*P-100*Q-10*R)
320 D(1)=P
330 D(2)=Q
340 D(3)=R
350 D(4)=S
360 FOR E=1 TO 4
370 IF D(E)<>B(E) THEN 420
380 PRINT " BLACK";
390 B(E)=B(E)+10
400 D(E)=D(E)+20
410 H=H+1
420 NEXT E
430 IF H=4 THEN 630
440 FOR F=1 TO 4
450 D=D(F)
460 FOR G=1 TO 4
```

```
470 IF D<>B(G) THEN 510
480 PRINT " WHITE";
490 B(G)=B(G)+10
500 GOTO 520
510 NEXT G
520 NEXT F
530 FOR G=1 TO 4
540 IF B(G)<10 THEN 560
550 B(G)=B(G)-10
560 NEXT G
570 H=0
580 PRINT
590 NEXT C
600 PRINT:PRINT:PRINT"You didn't get it.
..."
610 PRINT:PRINT"The answer was ";B(1);B(
2);B(3);B(4)
620 END
630 PRINT:PRINT:PRINT"Well done, Codebre
aker."
640 PRINT:PRINT"You got the answer in ju
st ";C;"goes"
```

Here's what the screen looks like after one round:

```
Enter guess number 7
9385  BLACK BLACK WHITE
Enter guess number 8
9357  BLACK BLACK BLACK BLACK

Well done, Codebreaker.
```

```
You got the answer in just 8 goes
```

We'll now go through the program line by line, a practice we'll be following for several of the programs in this book. If you don't want to read the detailed explanation now (and there may be parts of it which are a little bit difficult to understand at your present stage), by all means skip over the explanation and then come back to it later when you know a little more.

Lines 1Ø and 5Ø print a number of asterisks to rule off the title, with blank lines printed by lines 2Ø and 4Ø.

Line 14Ø pauses for a few seconds to allow you to read the instructions. Two *arrays* are dimensioned in lines 16Ø and 17Ø. We get to arrays in a later chapter. For now, all you need to know is that by saying DIM B(4) you tell the computer you want to create a *list* of objects, called B, in which the first item can be referred to as B(1), the second as B(2), and so on. These two arrays are used for storing the numbers picked up by the computer, and for your guess.

H is a *numeric variable* (we've mentioned numeric variables before) which is set equal to zero in this line. In line 41Ø, one is added to the value of H each time a black is found, so that if H equals four, the computer knows all the digits have been guessed and goes to the routine form line 63Ø to print up the congratulations.

The lines from 19Ø to 21Ø work out the number which you will have to try and guess. Line 2ØØ uses the RND function we've talked about to get four random numbers between zero and nine, and stores one each in the elements of the B array. Note that the first FOR/NEXT loop of our program appears here. The A in line 19Ø equals one the first time the loop is passed through, two the second time, and so on, so that the A in line 2ØØ changes as well.

Our next FOR/NEXT loop, which uses C, starts in the next line. It counts from one to eight, to give you eight guesses. Line 23Ø accepts your guess, using the words 'Enter guess number 1' as an input prompt. The numeric variable X is et equal to your guess, and line 24Ø checks to make sure you have not entered a five-digit number. If you have (if X is greater than 9999) then the program goes back to line 23Ø to accept another guess.

The next section of the program, right through to line 58Ø, works out how well you have done, using a number of FOR/NEXT loops (36Ø to 42Ø, 44Ø to 52Ø, 46Ø to 51Ø and 53Ø to 56Ø). Line 59Ø sends the program back to the line after the original FOR C = . . . to go through the loop again. If the C loop has been run through eight times, then the program does not go back to line 23Ø, but 'falls through' line 59Ø to tell you that you have not guessed the code in the eight guesses you were given, and to tell you what it is. Line 61Ø prints out the code.

If you do manage to guess the correct code, so that H equals four in line 43Ø, then the program jumps to line 63Ø to print out the congratulatory message.

PACKING 'EM IN

You can save quite a lot of space in the program (although it doesn't always make easier to work out what is going on within the program, or pick up errors) by using *multi-statement lines.* With a multi-statement line, you put more than one statement to each line number. Each statement on the same line must be separated by a colon (:).

Here is a slightly condensed version of CODEBREAKER produced by using multi-statement lines. Compare the two listings, and see how a little bit of space can be saved with multi-statement lines. Do *NOT* put a second statement in a line after the word IF (such as line 43Ø in the original listing). The reason for this will be outlined in the next chapter.

```
10 CLS:PRINT "**************************
************"
20 PRINT:PRINT"CODEBREAKER BY A.J.B, T.S
. AND P.J.S. ":PRINT
30 PRINT"********************************
*******"
40 PRINT "When you are told to do so, en
ter a"
50 PRINT "four-digit number and then pre
ss"
60 PRINT"RETURN."
70 PRINT"Digits can be repeated."
80 PRINT"You have 8 goes to break the co
de!"
90 PRINT"--------------------------------
-------"
100 WAIT 350:CLS:DIM B(4):DIM D(4):H=0
110 FOR A=1 TO 4:B(A)=INT (RND(1)*10):NE
XT A
120 FOR C=1 TO 8:PRINT "Enter guess numb
er ";C;
130 INPUT X:IF X>9999 THEN 130
140 PRINT:IF INT(X/1000)=0 THEN PRINT"0"
;
150 PRINTX;:P=INT(X/1000):Q=INT((X-1000*
P)/100)
160 R=INT((X-1000*P-100*Q)/10):S=(X-1000
*P-100*Q-10*R)
170 D(1)=P:D(2)=Q:D(3)=R:D(4)=S
180 FOR E=1 TO 4:IF D(E)<>B(E) THEN 200
190 PRINT" BLACK";:B(E)=B(E)+10:D(E)=D(E
```

```
)+20:H=H+1
200 NEXT E:IF H=4 THEN 320
210 FOR F=1 TO 4:D=D(F)
220 FOR G=1 TO 4:IF D<>B(G) THEN 240
230 PRINT" WHITE";:B(G)=B(G)+10:GOTO 250
240 NEXT G
250 NEXT F
260 FOR G=1 TO 4:IF B(G)<10 THEN 280
270 B(G)=B(G)-10
280 NEXT G:H=0:PRINT:NEXT C:PRINT
290 PRINT:PRINT"You didn't get it..."
300 PRINT:PRINT"The answer was ";B(1);B(
2);B(3);B(4)
310 END
320 PRINT:PRINT:PRINT"Well done, Codebre
aker."
330 PRINT:PRINT"You got the answer in ju
st ";C;"goes"
```

CHAPTER SIX
CHANGING IN MID-STREAM

We pointed out earlier in the book that, in most cases the computer follows through a program in line order, starting at the lowest line number and following through in order until the program reached the full line.

This is not always true. The GOTO command sends action through a program in which ever order you decide.

Enter the following program, and *before* you run it, see if you can work out what will appear on the screen:

```
5 TRON
10 GOTO 40
20 PRINT"THIS IS 20"
30 GOTO 60
40 PRINT"  THIS IS 40"
50 GOTO 20
60 PRINT"     THIS IS 60"
70 GOTO 40
```

This rather pointless program sends the poor computer jumping all over the place, changing its position in the program every time it comes to a GOTO command. Here's what you should see on your screen:

```
[70][40]  THIS IS 40
[50][20]THIS IS 20
[30][60]    THIS IS 60
[70][40]  THIS IS 40
[50][20]THIS IS 20
[30][60]    THIS IS 60
[70][40]  THIS IS 40
[50][20]THIS IS 20
[30][60]    THIS IS 60
[70][40]  THIS IS 40
[50][20]THIS IS 20
[30][60]    THIS IS 60
[70][40]  THIS IS 40
[50][20]THIS IS 20
```

Look at line 5;

```
5 TRON
```

Now you may wonder what that's doing there, well it stands for TRace ON, that is the number printed in the square brackets is equal to the current line number. To switch it off you use TROFF. When you get round to writing your own programs, the TRON command may come in very useful when you're not sure what the computer is doing. The TRON instruction allows you to trace the line number.

When the computer reaches line 10 it finds a GOTO command and moves onto line 40. At line 40 the computer prints some information and then comes to another GOTO command in line 50. From here it goes to 20, prints and then goes to 60. The Computer is then sent back to line 40 and the circle is complete. It is highly unlikely that you will ever use a program that is that short with so many jumps to any advantage.

RESTRICTIVE PRACTICES

Using GOTO In this way is called *unconditional.* The command is not qualified in any way, so the computer always obeys it. Another pair of words generally found together in a program are IF and THEN. IF something is true, THEN do something. The next program, which 'rolls a die' (using the random number generator) and then prints up the result of that die roll as a word, uses a number of IF/ THEN lines:

```
10 REM DICE ROLLS
20 GOTO 100
30 PRINT"ONE":GOTO 100
40 PRINT"TWO":GOTO 100
50 PRINT"THREE":GOTO 100
60 PRINT"FOUR":GOTO 100
70 PRINT"FIVE":GOTO 100
80 PRINT"SIX"
100 A=INT(RND(1)*6)+1
110 IF A=1 THEN GOTO 30
120 IF A=2 THEN GOTO 40
130 IF A=3 THEN GOTO 50
140 IF A=4 THEN GOTO 60
150 IF A=5 THEN GOTO 70
160 IF A=6 THEN GOTO 80
```

This is what you'll see when you run this program:

```
SIX
FIVE
TWO
FOUR
THREE
TWO
ONE
```

```
ONE
FOUR
FIVE
```

So we've looked at unconditional, and conditional GOTOs to send action all about the place within a program.

There is another way to redirect the computer during the course of a program by the use of *subroutines*. A subroutine is part of a program which can be called any number of times during a program, and is more efficiently kept outside the main program, than within it.

This example should make it clear. In this, the computer throws a die over and over again. The first time it is thrown the computer is throwing it for you. The second time it is thrown it is throwing the die for itself. After each pair of throws the computer will announce the winner (highest number wins). The main part of this program, that we want you to grasp the idea of, is the subroutine used to throw the die, this is done by first telling the computer to GOSUB, (GO to SUBroutine), and at the end of the subroutine there is a RETURN statement which tells the computer to go back to where it came from in the main program and carry on with the next command after the GOSUB. Enter and run the program, then return to the book and we'll explain where the subroutine is within the program, and how it works:

```
10 WAIT 50
20 FOR C=1 TO 2
30 GOSUB 100
40 IF C=1 THEN A=D
50 IF C=2 THEN B=D
60 NEXT C
70 IF A>B THEN PRINT"I WIN" ELSE PRINT"Y
```

```
OU WIN"
80 GOTO 10
100 REM THIS IS THE SUBROUTINE
110 D=INT(RND(1)*6)+1
120 IF C=1 THEN PRINT"I ROLLED A ";D;" "
 ELSE PRINT "YOU ROLLED A " ;D
130 WAIT 30
140 RETURN
```

This is what you'll see when you run it:

```
I ROLLED A 4
YOU ROLLED A 3
I WIN
I ROLLED A 5
YOU ROLLED A 1
I WIN
I ROLLED A 4
YOU ROLLED A 5
YOU WIN
```

In line 10 you'll notice a new instruction, WAIT. In the flashing TESTING program earlier in the book we used a FOR/NEXT loop to create a small pause, fortunately Oric BASIC allows you to have the small pause without the need to use FOR/NEXT loops, this instruction is the WAIT statement. The number following the WAIT statement obviously determines how long the pause will last, experiment with WAIT.

Line 20 contains the ominous FOR command, in this case it is used to determine whether it's the computer's throw or yours.

When it gets to line 30, which it does (of course) once each time through the C loop, the program is sent to the *subroutine*

starting at line 100. The die is 'rolled' in line 110, and the numeric variable D is set equal to the result.

In the next line we introduce a new twist to the usual IF..THEN statement, from here on in you will have to refer to IF..THEN statements as IF..THEN..ELSE, this is because you can, with Oric BASIC, use a command called ELSE, which will carry out the instructions following the ELSE command if the expression following the IF statement is false, i.e. the command:

IF 1=2 THEN PRINT "1=2" ELSE PRINT "1 does not equal 2" will obviously print "1 does not equal 2" because the expression '1=2' is false, so the instructions after the ELSE command are followed.

There is another WAIT statement following the IF..THEN..ELSE command which provides a short pause for you to read the information on the screen.

In the final line of the subroutine there is the RETURN command, which, as we explained before, returns the action to the main program, i.e. line 40. There, the IF/THENs in lines 40 and 50 determine whether the value of the roll (D) should be assigned to the variable A or B.

Line 60 ends the FOR/NEXT loop, and then line 70 determines whether you or the computer has won, using the ELSE command within the IF..THEN..ELSE. From here the computer GOTOs line 10 and the program starts again.

Study this example until you're sure you know how subroutines work, because they're pretty useful things in nearly every program.

LET'S ROLL AGAIN

There is a very efficient way of shortening the previous die rolling program using the command ON..GOSUB, (there is also a command ON.. GOTO which works in a similar way), to see the effect of ON..GOSUB type in the following program and run it:

```
10 REM DICE ROLLER
20 D=INT (RND(1)*6)+1
30 ON D GOSUB 50,60,70,80,90,100
40 GOTO 20
50 PRINT"ONE":RETURN
60 PRINT"TWO":RETURN
70 PRINT"THREE":RETURN
80 PRINT"FOUR":RETURN
90 PRINT"FIVE":RETURN
100 PRINT"SIX":RETURN
```

As you can see, the effect of ON..GOSUB, or ON.. GOTO for that matter, is that the computer goes to the subroutine Nth in the list in the expression ON N GOSUB A,B,C etc., i.e. if N = 3 and C = 100 then the computer will go to the subroutine at line 100.

CHAPTER SEVEN
SOUNDING OUT

The Oric 1 microcomputer is supplied with a very useful set of sound commands which will enhance your programs no end.

The four most noticeable commands available are the predefined sound commands EXPLODE, PING, SHOOT and ZAP. These are extremely useful if envelopes, tone and noise and waveforms have you baffled. To see what each sounds like, type in the following program and run it:

```
10 REM SOUND DEMO 1
20 CLS
30 PRINT"EXPLODE"
40 EXPLODE:WAIT 100
50 CLS
60 PRINT"PING"
70 PING:WAIT 50
80 CLS
90 PRINT"SHOOT"
100 SHOOT:WAIT 50
110 CLS
120 PRINT"ZAP"
130 ZAP:WAIT 30
140 GOTO 20
```

Now, armed with these four new commands, you can liberally spread them throughout the programs and program fragments we showed you earlier in this book.

SELF COMPOSURE

If you decide that you want a little more than just explosions, etc., then you can create your own sounds using the commands MUSIC, SOUND and PLAY.

MUSIC

The music command offers pure notes, so therefore is aimed at the musicians amongst you. The correct syntax for MUSIC is;

MUSIC (Channel, Octave, Note, Volume)

Where Channel is 1, 2 or 3 tone channels Octave — Ø to 6, Ø being the lowest and Note:

1 = C
2 = C#
3 = D
4 = D#
5 = E
6 = F
7 = F#
8 = G
9 = G#
1Ø = A
11 = A#
12 = B

SOUND

Sound is a command more useful when you want to create effects, i.e. wind or rain. The correct syntax for SOUND is;

SOUND (Channel, Period, Volume)

Channel: 1, 2 or 3 tone channels
4, 5 or 6 noise channels

PERIOD: Sound length

VOLUME: Ø–15 if Ø is used, as with MUSIC command, control is taken by the envelope.

PLAY

Play is the most useful of all the sound commands when it comes to creating synthesised effects. Its parameters are:

PLAY (Tone enable, Noise enable, Envelope Mode, Period)

Tone enable: Ø No tone channels on
1 Channel 1 on
2 Channel 2 on
3 Channels 1 and 2 on
4 Channel 3 on
5 Channels 3 and 1 on
6 Channels 3 and 2 on
7 Channels 3, 2 and 1 on

Noise enable: as above, only for noise channels

Envelope mode: 1 Single decay.
2 Single rise, then silent.
3 continuous sawtooth, starting from a peak
4 continuous rise and fall, starting low.
5 curved rise with continuous high tone.
6 continuous rising sawtooth.
7 straight rise to continuous high tone.

PERIOD: Ø to 32767

Try the following program which uses all of Oric's sound facilities to show you their possibilities.

```
10 REM SOUND DEMO 2
20 PRINT CHR$(6):CLS
30 PRINT"BOMBS AWAY"
40 FOR A=6 TO 0 STEP -1
50 :FOR B=12 TO 1 STEP -1
60 : MUSIC 1,A,B,15
70 :NEXT B
80 NEXT A
90 EXPLODE:WAIT 100
100 CLS
110 PRINT"HELICOPTERS"
120 :PLAY 0,2,3,1
130 WAIT 200
140 CLS:PRINT"INVADERS"
150 FOR A=1 TO 5
160 :WAIT INT(RND(1)*30)+20
170 :ZAP
180 NEXT A
190 CLS:PRINT"MUSIC"
200 RESTORE
210 FOR A=1 TO 23
230 :READ N,P
240 :MUSIC 1,2,N,0
250 :PLAY 1,0,2,100:WAIT 2:PLAY 1,0,1,30
00
260 :WAIT P*30:IF A>15 THEN WAIT 10
270 :PLAY 0,0,0,0
280 NEXT A
290 DATA 10,3,10,3,8,.5,10,.5,12,4
300 DATA 8,.5,10,.5,12,4,10,.5,8,.5
```

```
310 DATA 10,5,8,.5,7,.5,7,5,5,3.5,3,.5
320 DATA 5,3,3,2.5,3,.5,3,4,5,1.5,2,.5,3
,5
330 PING:WAIT 50:EXPLODE
340 WAIT 100:CLS
350 PRINT"GUNFIGHT"
360 FOR A=1 TO 5
370 :WAIT INT(RND(1)*30)+20
380 :SHOOT
390 NEXT A
400 SOUND 1,1,1
410 PLAY 0,0,0,0
```

THE DESCRIPTION

Line 10 holds that famous REM statement which is here holding the name of the program *SOUND DEMO 2*. In line 20 the effect of PRINTing CHR$(6) is to switch off the keyboard click, which would otherwise effect the RUNning of this program. Line 30 gives us a clue as to what sort of effect we are trying to get in lines 40–90, listen to the noise and refer to the program listing, can you see what the computer is doing? From line 110 to line 130 you will find the routine which tries to recreate the sound of a helicopter by quickly switching on and off the noise channel via the envelope. The *INVADERS* sounds are simply ZAP commands at random intervals. From line 190 to line 320 you will see the MUSIC routine, which uses a few commands we haven't covered yet, but we'll get round to it. All you need to know now is that DATA holds the note values and lengths, RESTORE makes sure that the computer will look at the information from line 290 onwards and the READ command takes the information out of the DATA statements. The result is a tune, although not everybody's favourite, is a bit more complicated than ZAP. *GUNFIGHT* from line 350 is similar to *INVADERS* but uses the SHOOT command. In line 410 switches off any sounds that your Oric might still be making.

SOUND ADVICE

As you will have noticed in the Sound Demo 2 program, you can use variables within the sound commands SOUND, MUSIC and PLAY. In the next program, SOUND ADVICE, you have to guess the number between 1 and 12. The feedback on each guess — which will help you home in on the correct number in the shortest possible number of guesses — is in the form of MUSIC. Once you've played it for a few rounds, you'll learn to interpret the output from the computer.

When you manage to guess the correct answer, you'll get this on the screen:

```
Yes, it was 11
You got it in 3  guesses
```

Here's the listing:

```
10 REM SOUND ADVICE
20 R=INT (RND(1)*12)+1
30 PAPER 1:INK 7:CLS
40 GUESS=1
50 PLOT10,9,"THIS IS GUESS #"+STR$(GUESS
)+"          "
60 FOR A=1 TO 20:PRINT:NEXT:INPUT "What
number am I thinking of";A:
CLS
80 IF A=R THEN 130
90 PLOT 4,5,"No, your guess is not corre
ct"
100 MUSIC 1,3,R,15:WAIT 50:PLAY 0,0,0,0:
CLS
110 GUESS=GUESS+1
```

```
120 GOTO 50
130 CLS:PRINT:PRINT:PRINT"Yes, it was ";
R
140 PRINT:PRINT:PRINT"You got it in ";GU
ESS;" guesses"
```

First of all is the REM statement telling us the name of the program. The next line generates a random number between 1 and 12, and sets the numeric variable R to it. In line 30, the computer sets the background (PAPER), to red, and the colour in which it prints (INK), to white. Both PAPER & INK are automatically established, this will be explained in the colour chapter of this book.

Line 40 sets the numeric variable GUESS to one, and in line 50 the number is *PLOTed* onto the screen. Using PLOT you can specify where you want the information on the screen, i.e. PLOT 0,0,"TOP CORNER".

PLOTs the string "TOP CORNER" in the top left-hand corner of the screen. If you were to say PLOT 12,12,"CENTER" then the computer would print the string "CENTER" in the middle of the screen. Line 60 accepts your guess, as variable A. If A (the number you've guessed) is the same as R) the number the computer thought of in line 20), as the computer checks in line 80, then the computer knows you have guessed its number.

If you are not right then line 90 tells you so, and the following line (100) gives you your audio feedback, the pitch will give you the clue as to what it is. The variable GUESS is *incremented* by

one (that is, one is added to the value of the variable GUESS) before the computer returns to line 5Ø to accept your next guess.

If you have guessed the number correctly (detected in line 8Ø) the program moves to line 13Ø. The screen is cleared and the computer tells you the number it was thinking of, and then (in line 14Ø) tells you how many guesses you took.

```
120 GOTO 50
130 CLS:PRINT:PRINT:PRINT"Yes, it was ";
R
140 PRINT:PRINT:PRINT"You got it in ";GU
ESS;" guesses"
```

First of all is the REM statement telling us the name of the program. The next line generates a random number between 1 and 12, and sets the numeric variable R to it. In line 30, the computer sets the background (PAPER), to red, and the colour in which it prints (INK), to white. Both PAPER & INK are automatically established, this will be explained in the colour chapter of this book.

Line 40 sets the numeric variable GUESS to one, and in line 50 the number is *PLOTed* onto the screen. Using PLOT you can specify where you want the information on the screen, i.e. PLOT 0,0,"TOP CORNER".

PLOTs the string "TOP CORNER" in the top left-hand corner of the screen. If you were to say PLOT 12,12,"CENTER" then the computer would print the string "CENTER" in the middle of the screen. Line 60 accepts your guess, as variable A. If A (the number you've guessed) is the same as R) the number the computer thought of in line 20), as the computer checks in line 80, then the computer knows you have guessed its number.

If you are not right then line 90 tells you so, and the following line (100) gives you your audio feedback, the pitch will give you the clue as to what it is. The variable GUESS is *incremented* by

one (that is, one is added to the value of the variable GUESS) before the computer returns to line 5Ø to accept your next guess.

If you have guessed the number correctly (detected in line 8Ø) the program moves to line 13Ø. The screen is cleared and the computer tells you the number it was thinking of, and then (in line 14Ø) tells you how many guesses you took.

CHAPTER EIGHT
MAKING COMPARISONS

We all know the equals sign (=) and we've seen it in use in several programs so far. We've also seen the greater than (>) and less than (<) signs. At this point of your learning, we thought it would be useful to briefly recap on what each of these signs are, and what they mean:

= equals
> greater than
< less than
>= greater than or equal to
<= less than or equal to
<> not equal to

You'll see these in use in many programs in this book, such as this next one, which allows you to challenge the computer to a game of BRICKUP, written by Graham Charlton.

Here's the listing for BRICKUP:

```
10 REM BRICKUP
20 REM From a program written by
30 REM Graham Charlton
40 GOSUB 8000
50 T=10:SC=0
60 PLOT 1,0,"LIVES LEFT "+STR$(T):PLOT 1
9,0,"SCORE "+STR$(SC)
```

```
70 C=1:D=1:X=23:Y=INT(RND(1)*10)+20:M=Y-
2
80 IFSCRN(Y,X)=ASC("X")THENC=-C:SC=SC+1:
SOUND1,50,15:PLOT 25,0,STR$
(SC)" "
90 IF X>24 THEN 1000
100 PLOT Y,X,"O"
110 PLOT M,24," [==] "
120 IF KEY$="P" THEN M=M+1
121 WAIT 5:PLAY 0,0,0,0
122 IF KEY$"E" THEN M=M-1
130 IF X+C<3 THEN C=-C:SOUND 1,50,15
140 IF X=24 AND Y>=M AND Y<=M+5 THEN C=-
C:SOUND 1,40,15
150 IF Y+D<2 OR Y+D>35 THEN D=-D:SOUND 1
,60,15
160 PLOT Y,X," "
170 X=X+C:Y=Y+D
180 GOTO 80
1000 T=T-1:C=-C:MUSIC1,0,2,15:WAIT50:MUS
IC1,0,1,15:WAIT 100:PLAY 0,
0,0,0
1010 PLOT 1,0,"LIVES LEFT "+STR$(T)+" ":
PLOT 19,0,"SCORE "+STR$(SC)
1020 PLOT M,24,"       "
1030 IF T>0 THEN 70
1040 EXPLODE:CLS:END
8000 CLS:PAPER 4:INK 7
8010 PRINT:FOR A=1 TO 36:PINT CHR$(126);
:NEXT
8020 FOR A=2 TO 25:PLOT 1,A,CHR$(126):PL
```

```
OT 36,A,CHR$(126):NEXT A
8030 PLOT 2,5,"XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXX"
8040 PLOT 2,6,"XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXX"
8050 PLOT 2,7,"XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXX"
8060 PLOT 2,8,"XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXX"
8070 PRINTCHR$(17);
8080 RETURN
```

In BRICKBAT, you use the P and E keys to move the little slide at the bottom of the screen back and forth, bouncing the ball off it (if you can). As the ball bounces up to the X's above, it will strike them, making a sound and adding to your score. You have ten balls per round. It is fascinating to see what happens if the ball gets momentarily 'trapped' *inside* the bricks, bouncing off them wildly, before it finds a way out and down. You'll see how effective this can be when you run the program. The only limitation of this program is the way the Oric reads the keyboard, causing slow reactions to your commands.

Notice that our 'comparison symbols' (less than, and the rest) are used frequently in this program. They perform a number of tests in conjunction with IF/THEN statements, bouncing the ball off the slide at the bottom of the screen, the walls or the bricks, deciding when the ball has missed the slide, and continuing the game if you still have a 'life' (that is, ball) left. The program ends in line 1040 with an explosion.

The words AND and OR are also used in comparison lines, *chaining* two or more tests together, as in lines 140 and 150. They work as follows:

AND The computer does what follows the THEN if *both* of the conditions chained by the AND are true

OR The computer carries out the instruction following the THEN if *either* of the conditions are true.

CHAPTER NINE
A GAME AND A SPEED TEST

It's time now to take a break from the serious business of learning to program the computer. As you can see in this chapter, we have a couple of major programs, which use many commands that have not yet been explained. We suggest you enter the programs just as they are, play them for your own enjoyment, then come back to the explanations which follow the listings *after* you have mastered the rest of the book. We do not think it is fair to keep you waiting for major programs until you've covered everything on the computer, so we have decided to supply you with these programs at this point, and hope you'll enter them 'on trust', returning to this chapter for the explanations when you feel you are ready. Of course, you do not have to enter the programs right now. If you'd prefer to continue with the learning, then move straight along to chapter ten.

PLAYING ALONE

Our first listing allows you to use your computer as a Solitaire board. It is believed that Solitaire was invented by an imprisoned nobleman in France in the late 16th century, and was brought into England in the closing decade of the 18th century.

The aim of the game is simple to explain, but not so easy to achieve. You start on a board which has 33 positions marked.

There are 'pegs' in 32 of the positions on a real board, and the middle position is empty.

The 'blocks' represent the pegs, and the white mark in the center is the empty square. To play, you simply jump over any of your pieces vertically or horizontally, so that you end up in an empty position. The piece which you have jumped over is removed from the board. The aim of the game is to end up with just one block in the center position.

The board is displayed each move, and also the number of pegs, (blocks), left on the screen. You move by entering the co-ordinates of the piece you want to move, using the number down the side first, followed by the number across the top. These are entered as a single, double-digit number. If you wanted to move the piece which was two positions below the central hole, you'd enter 64, then press RETURN, followed by 44, and RETURN again. The board is reprinted, and you are then offered another move.

This is the listing of the Solitaire program:

```
10 REM SOLITAIRE
20 GOSUB 260
40 REM ACCEPT MOVE
45 CLS:GOSUB 190
50 INPUT "Which peg do you wish to move"
;A$
60 MUSIC 1,1,1,15:WAIT 5:PLAY 0,0,0,0
70 LET A1=VAL(LEFT$(A$,1)):A2=VAL(RIGHT$
(A$,1))
75 IF A(A1,A2)<>MAN THEN GOTO 50
80 INPUT "to where";B$
90 MUSIC 1,1,1,15:WAIT 5:PLAY 0,0,0,0
93 LET B1=VA(LEFT$(B$,1)):B2=VAL(RIGHT$(
```

```
B$,1))
95 IF A(B1,B2)<>EMPTY THEN 80
100 IF ABS(A1-B1)>2 OR ABS(A2-B2)>2 THEN
 45
101 IF ABS(A1-B1)=1 OR ABS(A2-B2)=1 THEN
 45
102 IF ABS(A1-B1)=0 AND ABS(A2-B2)=0 THE
N 45
103 IF ABS(A1-B1)=2 AND ABS(A2-B2)=2 THE
N 45
110 MOVE=MOVE+1:COUNT=0
115 A(A1,A2)=EMPTY:A(B1,B2)=MAN
116 A((A1+B1)/2,(A2+B2)/2)=EMPTY
120 FOR L1=1 TO 7
125 FOR L2=1 TO 7
130 IF A(L1,L2)=MAN THEN COUNT=COUNT+1
135 NEXT L2
140 NEXT L1
160 IFCOUNT=1ANDA(4,4)=MANTHENPRINT"YOU
DID IT IN "MOVE:END
170 PLOT1,8,"THERE ARE "+STR$(COUNT)+"PE
GS ON THE BOARD"
180 GOTO 40
190 PLOT 16,5,"1234567":PLOT 16,6,"!!!!!
!!"
200 FOR LOOP=-7 TO -1:PLOT 24,6+ABS(LOOP
),STR$(LOOP):NEXT LOOP
210 FOR L1=1 TO 7:FOR L2=1 TO 7
220 PLOT 15+L2,6+L1,CHR$(A(L1,L2))
230 NEXT L2:NEXT L1
240 PLOT 17,7,16:PLOT 17,8,16:PLOT 15,9,
```

```
16:PLOT 15,10,16
241 PLOT 17,14,16:PLOT 21,14,20
245 PLOT 21,7,20:PLOT 21,8,20:PLOT 23,9,
20:PLOT 23,10,20
246 PLOT23,11,20:PLOT 23,12,20:PLOT 21,1
3,20
250 PLOT 15,11,16:PLOT 15,12,16:PLOT 17,
13,16:RETURN
260 PAPER 4:INK7:CLS:DIM A(7,7)
270 BLANK=32:MAN=33:EMPTY=34
280 FOR L1=1 TO 7:FOR L2=1 TO 7
290 READ A(L1,L2):NEXT L2,L1
300 FOR L1=46080+(MAN*8) TO 46080+(34*8)
+7
310 READ USER:POKE L1,USER:NEXT
320 RETURN
330 DATA 32,32,33,33,33,32,32
340 DATA 32,32,33,33,33,32,32
350 DATA 33,33,33,33,33,33,33
360 DATA 33,33,33,34,33,33,33
370 DATA 33,33,33,33,33,33,33
380 DATA 32,32,33,33,33,32,32
390 DATA 32,32,33,33,33,32,32
400 DATA 7,13,21,21,23,26,28,0
405 DATA 0,0,28,28,28,28,28,0
```

The program works as follows. Line 20 sends the action to the subroutine from line 260 which sets up the requirements for the start of the game. The PAPER is set to blue and the INK is set to

white. The array A is used to hold the board, as you can see, it has been dimensioned as A(7,7), this means that there are seven pigeonholes, with seven more sub-pigeonholes in each. More variables are initialised and then the pattern of the Solitaire board is read into A(). The FOR/NEXT loop which starts at 300 creates what's known as *USER-DEFINABLE GRAPHICS*, these will be covered in detail in Chapter 12.

When the computer RETURNs from line 320, it is immediately sent on another subroutine, this time starting at line 190. This subroutine prints the board, firstly the grid numbers, then the pegs and spaces and finally the colour is added using low-resolution PLOT, (see Chapter 14).

In line 70 the functions LEFT$ & RIGHT$ are used to split up the input A$. Using VAL we can turn a string variable into a numeric variable. Most of the rest of the program is basic maths to check if you have made an invalid move, etc.

TESTING YOUR SPEED

Our next program, REACTION TESTER, is much shorter than Solitaire, but just as much fun to play. You enter the program, type RUN, and the message 'STAND BY' appears. After an agonizing wait, STAND BY will vanish, to be replaced by a notice "OK, PRESS THE 'Z' KEY". As fast as you can, leap for the Z key, and press it, knowing that the computer is counting all the time.

The computer then tells you how quickly you reacted, and compares this with your previous best time. "BEST SO FAR IS..." appears on the screen, and the computer then waits for you to take your hands off the keyboard, to prevent cheating (as if you'd try to cheat on your computer), before the whole thing begins again. If you manage to get the reaction time below 5 then the program will stop and proclaim you the champ.

Here's the listing of REACTION TEST:

```
5 REM REACTION TEST
10 HI=1000:PAPER 0:INK 7
15 DIM B$(32):FOR A=1 TO 32:B$(A)=CHR$(1
26):NEXT
20 CLS:PLOT 5,5,"STAND BY
"
30 FOR A=1 TO 200+RND(1)*500
40 NEXT A
50 IF KEY$<>"" THEN GOTO 50
55 PLOT 5,5,"OK, PRESS THE 'Z' KEY"
60 COUNT=0
70 COUNT=COUNT+1
90 A$=KEY$
100 IF A$<>"Z" THEN 70
110 PRINT:PRINT:PRINT"SCORE:"COUNT"   "
120 IF COUNT<HI THEN HI=COUNT:MUSIC 1,5,
5,15:WAIT 50:PLAY 0,0,0,0
130 PLOT 0,0," BEST SO FAR IS "+STR$(HI)
:PLOT 0,1,CHR$(17)+B$
135 WAIT 100
136 IF KEY$<>"" THEN GOTO 136
140 IF HI<5 THEN 160
150 GOTO 20
160 REM
170 PRINT CHR$(12)
180 PRINT CHR$(4);CHR$(27);"N                YOU
 ARE THE CHAMP !!!"
185 PRINTCHR$(4)
190 END
```

Line 10 sets the Hiscore to 1000 and sets the colour of the paper to black (0) and the ink to white (7). Line 15 dimensions the

array B$ to hold 32 characters, and then fills the string with chequered characters (CHR$(126)). In line 20 we see the PLOT statement being used to print 'STAND BY' on the screen. Lines 30 and 40 provide a random pause, and line 50 checks to see whether you already have your finger on a key. Line 55 tells you it's alright to press the Z key and lines 60 to 100 do the counting. If you manage a new high score then the program celebrates with a short note. The loop continues at line 150 and the 'CHAMP' routine is from line 170 onwards. Notice the use of control characters, (PRINT CHR$(12), etc,) as used in the champ routine, to print double height, and flashing characters.

ADDING A LITTLE STRUCTURE

Now is as good a time as any to introduce you to the idea of structured programming. Although you may think at this stage of your learning, 'If-it-works-then-don't-start-fiddling-about-with-it', you will soon develop bad habits. One good way of writing programs is to put everything into short subroutine, which can be called when necessary. You will need to iron out all the '*bugs*' in your programs, (a bug is a problem in your program which makes the program do what you didn't expect it to do, the word bug comes from an accident with an IBM when a moth got inside the case, and while it was frying away within it managed to crash the system). The easiest way to find your bugs is to watch exactly what your computer is doing, then refer back to the program and find out why it's doing it. Structured programming makes things neat and tidy, and therefore easier to decipher what's going on. There are a few points to remember if you want to start 'structured' programming:

Try not to keep jumping all over the place using the command GOTO, this makes the flow of the program difficult to follow.

Always put an END command at the end of your program, if you want to end a program somewhere which is deeply nested in your program then use the STOP command.

Line numbers in steps of ten.

CHAPTER TEN
STRINGING ALONG

You'll recall that several times in this book so far we have referred to *numeric variables* (letters like A or B, words like COUNT and GUESS, and combinations such as K9 or M1) and to *string variables* (similar to above, but followed by the dollar ($) sign, i.e. A$, COUNT$ or K9$). In this chapter we'll be looking at strings, and at things you can do with them.

THE CHARACTER SET

Every letter, number or symbol the computer prints has an ASCII value. Telling the computer to print the *character* of that value produces the character. It is easy to understand this. The value of the letter "A" has nothing to do with the value assigned to A when it is a numeric variable, but refers to "A" when we want the computer to print it out. We'll put the "A" in quote marks when we refer to it as a letter. Try it now. Enter this into your computer and see what you get:

```
PRINT ASC("A")
```

You should get the answer 65. Now 65, is the ASCII, (ASC() stands for ASCII), of the letter "A". We can turn it back into an "A" by asking the computer to print the character which responds to ASCII code 65. We do this by using the function CHR$, which stands for Character-string.

You can get your computer to print out every ASCII code and character with the next short program, which has — as you can see — a very long printout.

```
10 REM SHOWING ASC() AND CHR$()
20 FOR A=32 TO 126
30 PRINT "ASCII "A,,,"CHARACTER "CHR$(A)
40 WAIT 5
50 PLAY 1,0,1,10
60 NEXT A
```

You'll notice that we don't print all the characters, this is because some of them are *CONTROL CHARACTERS* and produce very weird effects, (remember that little trick we learned earlier to clear the screen using CTRL & L, well that was a control character, try something like CTRL & G for a surprise). If you have a printer attached to your Oric, then you may like to keep a permanent record of what your computer can actually print, to do this change the word PRINT in line 30 to LPRINT, your printer might start printing strange Japanese characters, if it does this then look for a switch which will probably read 7-Bit ASCII/Special Characters (or something similar). Switch this to 7-Bit ASCII.

TESTING YOUR CHARACTER

Our next program is a reaction tester like the one you experienced earlier. However, you are not being tested just on speed. In this program, you have to find the *right key* on the keyboard as quickly as possible.

Make sure that you are in CAPS mode, i.e. the CAPS sign appears in the right-hand corner of the screen. A letter will appear in the middle of the screen. As quickly as you can, find that letter on the keyboard and press it. You will be told how long it took you, and this time compared with your best time.

Here is the listing:

```
10 REM KEYBOARD INSTRUCTOR
15 BEST=0:CLS:PAPER 0:INK 3
20 A=65+INT (RND(1)*26)
30 B=0:PRINT:PRINT:PRINT:PRINT:PRINT:PRI
NT:PRINT:PRINT
40 PLOT 16,5,CHR$(A)
50 A$=KEY$
60 IF A$<>"" THEN IF A$=CHR$(A) THEN 120
70 B=B+1
80 PLOT 15,10,STR$(B)
90 IF B<200 THEN 50
100 PRINT:PRINT:PRINT:PRINT:PRINT"SORRY,
 TIME IS UP"
110 GOTO 140
120 PRINT:PRINT:PRINT:PRINT:PRINT"WELL D
ONE, YOU SCORED"
130 PRINT200-B"ON THAT ONE"
140 IF 200-B>BEST THEN BEST=200-B
150 PRINT:PRINT"BEST TODAY:"BEST
160 ZAP:WAIT A*3:PLAY 0,1,1,100
170 CLS
180 GOTO 20
```

CUTTING THEM UP

One of the really great aspects of your computer is the way it can be used to manipulate strings. It is called 'string slicing' and works as follows. There are three commands which are usually used in string slicing, they are RIGHT$, LEFT$ and MID$. Let's say you have the string A$ which equals "HELLO" for instance, and you want the end two characters to equal B$, to do this you would say:

B$ = RIGHT$(A$,2)

What the function does is take the last 2 characters of A$ and put them into B$. A$ stays the same, but B$ = "LO". The function LEFT$ works in the same way, but obviously takes the characters from the start of the string (i.e. the left). To see how this works try this one, assuming A$ = "HELLO" still;

C$ = LEFT$(A$,3)

The string C$ should hold "HEL", to see if it is holding this type PRINT C$.

The third function used in string slicing is MID$, this is a very useful function as it allows you to take a character or a few characters from the middle of a string, or the characters from one point to the end of a string. Assuming that A$ has the same 'value', try:

D$ = MID$(A$,3,2)

What this does is take 2 characters from the string A$, starting at the third character. Now try this one:

E$ = MID$(A$,2)

This takes all the characters from the string A$ from the second character. i.e. E$ would equal "ELLO" and D$ equals "LL".

The next program takes the string "MATHEMATICS" and splits it up into many different ways.

```
10 A$="MATHEMATICS"
20 PRINT "A$ = "A$
30 B$=LEFT$(A$,4):PRINT B$
40 B$=RIGHT$(A$,4):PRINT B$
50 B$=MID$(A$,3,3):PRINT B$
60 B$=MID$(A$,6):PRINT B$
70 REM COULD THIS BE A NEW POP GROUP?
```

RUN the program and try to follow what it's doing as the

computer carries out each instruction. In line 10 the computer assigns A$ to the string "MATHEMATICS", and line 20 prints out A$. Line 30 takes out the word MATH and prints it on the screen. Line 40 takes out "TICS" and again prints it on the screen. The same is done in lines 50 and 60 where the words "THE and "MATICS" are printed. You can change the program to print other parts of A$, or you could even change the string A$, but be careful that when you are 'cutting' the string you are not trying to take parts of the string which are not there, i.e. if A$ = "ORIC" and you let B$ = MID$(A$,6,1) then Oric is not going to be too happy.

PUTTING THEM BACK TOGETHER

Strings can not only be cut up, but they can also be added together on your computer. The process of adding strings together is called the frightening-looking word *concatenation*. You can *concatenate* two complete strings together, or just add bits of them, as in our next program:

```
10 A$="FISH AND CHIPS"
20 B$="PEAS AND BEANS"
30 C$=A$+B$
40 PRINT"A$="A$
50 PRINT"B$="B$
60 PRINT"C$="C$
70 D=INT(RND(1)*14)+1
80 E=INT(RND(1)*14)+15
90 D$=MID$(C$,D,E-D)
100 PRINT "D$="D$
110 E$=A$+D$+B$
120 PRINT "E$="E$
```

Try and see what's being added to what in the listing before you run the program, can you see what the effect of adding A$ and B$ is in line 30?

You should get results like these:

```
A$=FISH AND CHIPS
B$=PEAS AND BEANS
C$=FISH AND CHIPSPEAS AND BEANS
D$=AND CHIPSPEAS
E$=FISH AND CHIPSAND CHIPSPEAS PEAS AND
BEANS

A$=FISH AND CHIPS
B$=PEAS AND BEANS
C$=FISH AND CHIPSPEAS AND BEANS
D$=AND CHIPSPEA
E$=FISH AND CHIPSAND CHIPSPEAPEAS AND BE
ANS
```

PLAYING AROUND

You can do a number of things with string slicing, as our next program demonstrates. NAME PYRAMID allows you to enter your name to produce a very interesting display. Once you've seen the program in action you'll understand why the program has been given the name it has.

```
1 REM  NAME PYRAMID
2 REM showing string slicing
5 PAPER 0:INK 7:CLS
10 INPUT "WHAT IS YOUR NAME ";A$
20 IF LEN(A$)>18 THEN A$=LEFT$(A$,15)
30 A=LEN(A$):REM 'LEN()' IS THEN LENGTH
OF THE STRING
40 PRINTTAB(30);"."
50 FOR G=1 TO A:PRINT TAB(12+(19-G));
55 PLOT 0,G,G/4+1
60 FOR H=1 TO 2*G:PRINT MID$(A$,G,1);
```

```
70 NEXT H
80 PRINT
90 PLAY 1,0,1,10
100 NEXT G
```

```
GG
EEEE
TTTTTT
TTTTTTTT
IIIIIIIIII
NNNNNNNNNNN
GGGGGGGGGGGGG

SSSSSSSSSSSSSSSS
TTTTTTTTTTTTTTTTTTT
AAAAAAAAAAAAAAAAAAAA
RRRRRRRRRRRRRRRRRRRRR
TTTTTTTTTTTTTTTTTTTTTTTTT
EEEEEEEEEEEEEEEEEEEEEEEEEE
DDDDDDDDDDDDDDDDDDDDDDDDDDD
```

PLAYING IT BACK

Our final program in this chapter shows one very effective use of string slicing, in which a string is progressively reduced by one element. When you run ECHO GULCH, you'll see a letter appear on the screen. It will then vanish. *Once it has vanished* (and not before, you will have a limited amount of time in which you must press the corresponding key).

If you have pressed the right key, you will hear the PING, and the letter will be replaced by a new one, this will stay on the screen for a shorter time. Each time a new letter appears, you will be given less time to see it, before you have to type it into the computer. If you make a mistake, the "SORRY, THAT'S

WRONG" message will appear, along with your score. If you manage to get all of the list of letters right, you'll be rewarded with a "YOU'RE THE CHAMP!!!" message. Here's the

```
10 REM Echo Gulch - showing string
       slicing
15 S=0:CLS
20 A$=MID$("IRFILGTLKTAFCIHSIOEAKCKSEWND
RULDHJRLDHJRLG",RND(1)*6+1)
30 PLOT 19,13,LEFT$(A$,1)
35 FOR G=1 TO 5*LEN(A$):NEXT G:CLS
40 LET B$=KEY$
50 IF B$<"A" OR B$>"Z" THEN GOTO 40
55 PLOT 19,13,B$
60 IFB$=LEFT$(A$,1) THEN FOR G=1 TO 10+R
ND(1)*30:NEXT G
65 PLOT 19,13,"*": FOR G=1 TO 100: NEXT
G
70 IF B$<>LEFT$(A$,1)THEN GOTO 110
80 A$=MID$(A$,2)
85 IF LEN(A$)=1 THEN GOTO 140
90 S=S+1
95 PING
100 GOTO 30
110 PRINT:PRINT:PRINT:PRINT:PRINT"SORRY,
 THAT'S WRONG "
115 EXPLODE
120 PRINT:PRINT"YOU SCORED ";S
130 END
140 PRINT:PRINT:PRINT"YOU'RE THE CHAMP !
! "
150 EXPLODE
```

HOW IT WORKS

The variable S, which holds your score, is set to zero in line 15, and line 2Ø sets the string, A$, to a long line of letters, starting from a random position 1–6 letters in. This method stops you learning the sequence easily, and it is also quick and efficient. Line 3Ø 'plots' the first element of A$ at 19, 13 and 35 waits for a while, to give you a chance to see the letter before the screen is cleared. The LEN() of a string is its *length*, that is the number of characters which make it up, i.e. the string "HELLO" would have a length of five, because there are five letters in the string. The short pause that line 35 creates is proportional to the number of characters left in the string A$. Therefore when you are doing well, and are getting a few correct, the time the letter is left on the screen decreases each turn.

KEY$

You may have already noticed the function KEY$ in programs before, and you may already have a good idea as to what it does. If you were to write a short program like, 1Ø PRINT KEY$:GOTO 1Ø, then run it, the effect would be to print the key you were pressing on the screen. KEY$ *reads* the keyboard to find out which key, if any, is being pressed. The advantage of KEY$ over INPUT is that there is no need to press RETURN, the disadvantage is that KEY$ can only hold one character at a time, which is most impractical when you want someone to enter their name. Another problem with KEY$ when writing fast moving programs is that it is very sluggish, and takes a long time to autorepeat, we will be learning a much quicker method later in the book, but it requires a greater knowledge of the machine to understand, and so for the moment we will stick to KEY$.

To get back to line 4Ø, the string B$ is set equal to KEY$, then in line 5Ø the computer checks to see if you have pressed a 'letter' key by finding out whether B$ is 'less than' "A", i.e. one of the characters from 32 to 64, or it is 'greater than' "Z", i.e. a character from 92 to 126. If B$ is anything but an uppercase letter, then it goes back to line 4Ø to get another letter. This

form of comparison of strings can be used to get things in alphabetical order ("ZEBRA" is *greater than* "AARDVARK", and "BEAST" is *less than* "CAMERA").

You can compare strings using equals (=) and 'anything but', or not equals (<>) as is shown in the next few lines of the program. Line 60 compares the key pressed (B$) with the first element of A$, and if they are the same, continues through the multi-statement line to create a pause, (for anticipation's sake), using the loop G. Line 65 prints an asterisk (*) on the spot where the next letter will appear. Line 70 compares B$ with the first element of A$ again and, if they are not equal, it sends the program to line 110, where you are told "SORRY, THAT'S WRONG", the computer then 'EXPLODE's', and your score is given. Line 80 strips the first character of A$ off, by using MID$(A$,2), and 85 checks to see if you've 'used up' A$, and if you have, then it sends the action to line 140, where you are proclaimed 'THE CHAMP'. If not, then line 90 increases your score, 95 'PINGs' and line 100 sends the computer back to line 30.

CHAPTER ELEVEN
READING DATA

In this chapter we'll be looking at three very useful additions to your programming vocabulary: READ, DATA and RESTORE. We have already come across these commands before, but they require a much more detailed explanation. They are used to get information *stored* in *one part* of the program to *another part* where it can be used.

Enter and run this program, which should make this a little clearer:

```
10 REM READ & DATA
30 FOR A=1 TO 10
40 READ B
50 PRINT B
60 NEXT A
70 DATA 6,3,84,532,2,543,24,1999,4,48598
```

Using line **40**, the program READs through the DATA statement in line **70** in order, printing up each item of DATA (with line **50**).

RESTORE moves the computer back to the *first* item of DATA in the program, as you'll discover if you modify the above program by adding line 55, so it reads as follows:

```
10 REM READ, DATA & RESTORE
30 FOR A=1 TO 10
40 READ B
50 PRINT B
55 RESTORE
60 NEXT A
70 DATA 6,3,84,532,2,543,24,1999,4,48598
```

It does *not* matter where in the program the DATA is stored. The computer will seek it out, in order from the first item of DATA in the program to the last, as our next program (which scatters the DATA about in an alarming way) convincingly demonstrates:

```
10 DATA 9:REM READ & DATA
20 DATA 4:FOR A=1 TO 5:DATA 898
30 READ B:DATA 73,4:PRINT B
40 NEXT A
```

READ and DATA work just as well with string information:

```
10 REM STRING READ & DATA
20 FOR A=1 TO 3:READ A$
30 PRINT A$:NEXT A
40 DATA ORIC,READS,STRING
```

You can mix numeric and string DATA within the same program, so long as you take care to ensure that when the program wants a numeric item that a number comes next in the program, and when it wants a string item, it finds it:

```
10 REM MIXED READ & DATA
20 READ A,B$,B,C$,A$,BK
30 PRINT A,B$,B,C$,A$,BK
40 DATA 4,ORIC,643,HARTNELL,SHAW,8783
```

In our next program, GALACTIC GROCER, written by Tim Rogers, the goodies you are trading with start off their lives in DATA statements. The relative value of these strange objects are stored in the two DATA statements which follow.

In this program you are the Galactic Grocer, trading such interstellar goodies as coal dust, reject soap, aspidistras and zinc hub-caps. As you'll discover, these items are in great demand throughout the galaxy. You start out with some money and a stock of goods, provided by Galactic Grocery Central, and you are attempting to zap around space, trading here and bartering there, to return to Central with more goods. On arrival, the goods in your hold will be sold for twice the amount you paid for them, and a tenth of the profits will go to you. The program is set over some 20 years, compressed into a quarter of an hour or so by the computer time continuum.

Out there, in the cut and thrust universe of interstellar commerce, you'll meet other hardnosed Grocers from other firms, planets and star systems. They'll be on their way to other points in the known universe, and will be willing to trade with you. Keep in mind — when considering purchasing anything from wily traders — that the point of the game, is to try and stockpile the really pricey merchandise.

```
100 PA=0
110 IN=6
120 GOSUB 7000
130 DEF FN R(X)=INT (RND(1)*X)+1
140 DEF FN S(X)=INT (RND(1)*X)+1
145 PRINT CHR$(12): REM CLS
150 PRINT CHR$(4);CHR$(27)"N"SPC(11)"GAL
```

```
ACTIC GROCER"
155 PRINT CHR$(4):REM SWITCH OFF DOUBLE
HIEGHT
900 X=0
910 I=PEEK(0)
1000 DIM C$(21)
1005 DIM O(21)
1010 DIM L(21)
1020 DIM M(21)
1025 RESTORE
1030 DIM P(21)
1040 FOR A=1 TO 21
1045 PLAY 1,1,1,10
1050 READ A$
1060 L(A)=LEN(A$)
1070 C$(A)=A$
1075 PLOT A,A,"*"
1080 NEXT A
1090 FOR A=1 TO 21
1100 IF A<6 THEN READ M(A):NEXT A
1110 M(A)=1
1120 NEXT A
1130 FOR A=1 TO 21
1135 PLAY 1,1,1,A
1140 READ P(A)
1150 IF RND(1)>.85 THEN O(A)=FN S(10000)
1155 PLOT 38-A,A,"*"
1160 NEXT A
1165 GOSUB 8500
1170 W=2100
1180 M=10000+FN S(10000)
```

```
1190 GOSUB 5000
1195 J1=J
2000 PA=0:IN=6:GOSUB 7000
2010 PRINT "* * G A L A C T I C  G R O
C E R * *"
2020 GOSUB 6000
2040 PRINT "Year ",,W
2050 PRINT "Money",,CHR$(95);M
2060 PRINT
2070 FOR A=1 TO 21
2080 IF O(A)<>0 THEN PRINT O(A),,C$(A)
2090 NEXT A
2100 B$="buy"
2110 GOSUB 3000
2130 B$="sell"
2140 GOSUB 3000
2150 W=W+1
2160 IF W<2120 THEN 2000
2170 GOSUB 5000
2180 J2=J
2200 PA=1:IN=7:GOSUB 7000
2210 PRINT "You have arrived in port."
2220 PRINT
2225 Z=J1:GOSUB 4500
2230 PRINT "You started out with "CHR$(9
5);Z$
2240 PRINT "worth of goods"
2245 Z=J2:GOSUB 4500
2250 PRINT
2260 PRINT "You now have "CHR$(95);Z$
2270 PRINT "Worth of goods"
2280 PRINT
```

```
2290 U=INT ((J2-J1)/J1*100)
2300 PRINT "You made "U"% profit"
2320 IF I<U THEN POKE 0,U:I=PEEK(0)
2330 PRINT
2340 PRINT "Best so far :"I"%"
2350 REM END OF GAME
2400 PLOT 7,26,"DO YOU WANT TO PLAY AGAI
N ?"
2405 Q$=KEY$:IF Q$<>"Y" AND Q$<>"N" THEN
 2405
2410 IF Q$="Y" THEN RUN
2420 END
3000 PRINT
3010 N=FN S(5)
3030 S$="buy"
3040 IF B$="buy" THEN S$="sell"
3045 PRINT "There are "N"traders wishing
 to"
3050 PRINT S$" some goods"
3055 PLOT 13,26,"PRESS ANY KEY"
3060 IF KEY$<>"" THEN 3060
3062 IF KEY$="" THEN 3062
3065 GOSUB 8000
3070 FOR A=1 TO N
3072 GOSUB 8000
3075 F=FN S(3)
3080 X=FN R(21)
3085 GOSUB 4000:IF X=0 THEN 3270
3090 Y=FN S(10000)
3095 P=FN S(Y)
3100 Z=FN R(100)
3110 GOSUB 6000
```

```
3140 Z=INT (Z*P(X))/100
3145 GOSUB 4500
3147 IF P>O(X)ANDB$="sell"THENPRINT"The
trader wants more ";
3148 IFP>O(X)ANDB$="sell"THENPRINT"than
you have.":GOTO3270
3150 PRINT "Trader "A"wishes to "S$
3160 PRINT P"to "Y;LEFT$(C$(X),L(X));" a
t"
3170 PRINT CHR$(95);Z$" each"
3180 PRINT
3190 PRINT "(You have "CHR$(95);M")"
3195 PRINT "(You have "O(X);LEFT$(C$(X),
L(X))")"
3200 GOSUB 7500
3201 IFZ*P>MTHENPRINT"You haven't enough
 money":GOTO3270
3205 PRINT "How many to "B$:INPUT U$
3210 REM
3215 U=VAL(U$)
3220 IF U<P OR U>Y THEN 3200
3225 IF B$="sell" THEN U=-U
3230 IF U<-O(X) THEN 3200
3240 IF M-U*Z<=0 THEN 3190
3250 M=M-U*Z
3260 O(X)=O(X)+U
3270 ZAP:WAIT 50:NEXT A
3275 IF N=0 THEN FOR H=1 TO 150:NEXT H
3280 RETURN
4000 REM
4005 PRINT "Trader "A"says:-"
4010 PRINT "'What do you want to "B$"?'"
```

```
4000 INPUT Z$
4025 IF Z$="0" THEN X=0:RETURN
4030 FOR B=1 TO 21
4040 IFZ$=MID$(C$(B),M(B)) THEN X=B:GOTO
 4100
4050 NEXT B
4060 PRINT
4080 GOTO 4020
4100 IF RND(1)<.8 THEN X=B:RETURN
4110 PRINT "Sorry, Trader "A"does not wa
nt to"
4115 PRINT S$" any "MID$(C$(B),M(B))
4120 X=0
4125 WAIT 50
4130 RETURN
4500 Z$=STR$(Z)
4510 FOR R=1 TO LEN(Z$)
4520 IF MID$(Z$,R,1)="." AND R=LEN(Z$) T
HEN RETURN
4530 IF MID$(Z$,R,1)="." AND R=LEN(Z$)-1
 THEN Z$=Z$+"0"
4540 NEXT R
4550 RETURN
5000 J=0
5010 FOR A=1 TO 21
5020 J=J+O(A)*P(A)*2
5030 NEXT A
5040 J=J+M
5050 RETURN
6000PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
6060 RETURN
```

```
7000 PAPER PA
7010 INK IN
7020 CLS
7030 RETURN
7500 NO=FN R(20)-FN R(20)
7510 MUSIC 1,2,1,0:PLAY 1,0,1,10
7520 WAIT 10
7530 MUSIC 1,3,1,0:PLAY 1,0,1,20
7540 WAIT 20
7550 RETURN
8000 IF S$="sell" THEN PA=5:IN=0:GOSUB 7
000:RETURN
8010 PA=1:IN=7:GOSUB 7000
8020 RETURN
8500 INK 0
8510 MUSIC 1,3,1,0
8520 PLAY 1,0,1,10
8530 WAIT 20
8540 INK 4
8550 MUSIC 1,1,1,0
8560 PLAY 1,0,1,30
8570 WAIT 30
8580 INK IN
8590 RETURN
9000 DATA TONNES COAL DUST,KG ORANGE PEE
L,GRAMS GRAIN DUST
9010 DATA GALLONS SILICON INK,BARS REJEC
T SOAP
9020 DATA GREEN SMARTIES,EX-DICTATORS,TE
ST-TUBES
9030 DATA TV SETS,GOLDFISH
```

```
9040 DATA MISSILES,ASPIDISTRAS,MATCH HEA
DS,SLIPPERS
9050 DATA CHIPPED COFFEE MUGS,ZINC HUB-C
APS,TOE NAILS
9060 DATA PEN TOPS,PAPER BAGS,BADGES,NOI
SES
9070 DATA 6,5,4,9,6
9080 DATA 10,10,3,30,.3,.1,15,1,250,3,10
00,4,.1,5,1.5,10,1
9090 DATA .2,.1,3,23
```

Arrays are dimensioned and variables are initialised in the lines 910 to 1160. The C$ array is filled with your initial supply of goods (the coal dust and the like) with the loop from 1040 to 1080, and a diagonal row of stars is printed (line 1075) to keep you amused while this is going on. Line 1150 ensures that around 15% of the 'O' array will contain numbers generated at random between one and ten thousand (using the 'S' function which was, you'll recall, defined in line 140). More stars are printed by line 1155.

Once the loop has been traversed, 1165 sends the program to the subroutine starting at 8500 which makes a few noises and flashes the INK colour a bit. Line 1170 sets the variable W (the year) to 2100, and 1180 sets up your starting money (variable

M) to a value somewhere between 1000 and 2000 (using function S again). Line 1190 sends the computer to the subroutine from line 5000, which works out the value of your cargo (J). Line 2000 assigns values to the PAPER and INK colours, then sends the action to the subroutine from line 7000 to put them in order.

The subroutine from line 6000 called from line 2020, prints out six blank lines before returning to lines 2040 and 2050 where the year (W) and your money (M) is printed out. Another blank line is printed (by line 2060) then the loop from 2070 to 2090

prints out your holdings. The string variable B$ is set to "buy" in line 2Ø1Ø, and then the subroutine from 3ØØØ (where the program is directed by line 211Ø) the entire buying and selling occurs. Before 3ØØØ is called again, by line 214Ø, B$ is set equal to "sell". The word assigned to B$ determines the word assigned to S$ (either "buy" in line 3Ø3Ø or "sell" in the following lines). Both B$ and S$ are used in the print lines from 315Ø, where you are told that "Trader 1 wishes to buy..." or Trader 1 wishes to sell...".

The year (W) is incremented in line 215Ø, and the next line checks to see if 2Ø years have elapsed since the game began. If they have (and W is not less than 212Ø, as checked by 216Ø) the program 'falls through' to the next line, where the final report is generated: "You have arrived in port...", and so on, until you are told how much profit you made (line 23ØØ). The 'best profit so far' is calculated, and 'POKEd' into location (Ø).

CHAPTER TWELVE
LIVING MORE COLOURFULLY

You have a colour computer on your hands, and so you should make the most of it. We'll start our investigation of ways to improve your programs by referring back to the FAST FOOD program which was first introduced in chapter four.

We have been using colour statements in many of the programs you've encountered so far, so you may have a pretty good idea of how to use the facilities. The Oric has two commonly used colour commands, INK and PAPER. INK, as you might expect, sets the foreground colour, i.e. the colour we want the computer to write in. PAPER, on the other hand, sets the background colour. To get the computer to print everything on a red background for example, type 'PAPER 1', if you want the foreground to be white then type 'INK 7', and so on.

Here's a version of the FAST FOOD program using INK & PAPER:

```
10 REM FAST FOOD WITH GLOBAL INK & PAPER
20 LET A=INT(RND(1)*4)+1
30 LET B=INT(RND(1)*7)
40 PAPER B:INK 7-B:IFB=8-B THEN INK 0
50 PRINT:PRINT"A IS"A" B IS "B
```

```
60 PRINT"YOU'VE ORDERED"
70 IF A=1 THEN PRINT"A HAMBURGER"
80 IF A=2 THEN PRINT"FRENCH FRIES"
90 IF A=3 THEN PRINT"SPARE RIBS"
100 IF A=4 THEN PRINT"A THICK SHAKE"
110 WAIT 50
120 GOTO 20
```

You'll notice that as soon as the PAPER or INK colour is called, the whole screen is effected. This is called Global colour commands.

Here is a list of the colour codes on the Oric:

0 Black
1 Red
2 Green
3 Yellow
4 Blue
5 Magenta
6 Cyan
7 White

If for, instance, you wanted a yellow background, then you would type PAPER 3, if you wanted a Blue ink to go with this lovely yellow background, then type INK 4.

You'll probably find that you will learn each colour's number, and the need for checking up will soon disappear.

You may be thinking, this is all very well but what if I want more than one paper colour, or more than one ink? This is possible, but not as easy to grasp.

The Oric has a character known as the escape character, there is even a button on the extreme left of the keyboard especially for it. It has a strange effect on otherwise normal letters, press the escape key now, followed by the Q key, surprised? You can also

include something like that inside the program by printing CHR$(27) which is the control character, followed by the character which corresponds to what you want. Here is a list of the characters, and their effect after a escape character has been printed.

@ INK BLACK
A INK RED
B INK GREEN
C INK YELLOW
D INK BLUE
E INK MAGENTA
F INK CYAN
G INK WHITE
H SINGLE HEIGHT-STEADY-STANDARD- CHARACTEI SET
I SINGLE HEIGHT-STEADY-ALTERNATE CHARACTE SET
J DOUBLE HEIGHT-STEADY-STANDARD CHARACTE SET
K DOUBLE HEIGHT-STEADY-ALTERNATE CHARACTER SET
L SINGLE HEIGHT-FLASHING-STANDARD CHARACTER SET
M SINGLE HEIGHT-FLASHING-ALTERNATE CHARACTER SET
N DOUBLE HEIGHT-FLASHING-STANDARD CHARACTER SET
O DOUBLE HEIGHT-FLASHING-ALTERNATE CHARACTER SET
P PAPER BLACK
Q PAPER RED
R PAPER GREEN
S PAPER YELLOW
T PAPER BLUE
U PAPER MAGENTA
V PAPER CYAN
W PAPER WHITE

Don't worry about those bits in the middle, we'll talk about

those later. The good bits are the characters from @–G and P–W. Try pressing the escape key, followed by one of those keys and see if the thing you expected to happen, did. Take note that the escape character only effects, at most, one line. You can put as many escape characters on that line, but they will not overlap. Try this new version of FAST FOOD using escape character colour:

```
10 REM FAST FOOD WITH COLOUR
15 PAPER 5:INK 0
20 A=INT(RND(1)*4)+1
30 B=INT(RND(1)*5)+1
40 PRINT:PRINT "A IS "A"   B IS "B
45 PRINT CHR$(27);CHR$(B+80);"YOU'VE ORD
ERED"
50 IF A=1 THEN PRINTCHR$(27);"RA HAMBURG
ER"
60 IF A=2 THEN PRINTCHR$(27);"SFRENCH FR
IES"
70 IF A=3 THEN PRINTCHR$(27);"TSPARE RIB
S"
80 IF A=4 THEN PRINTCHR$(27);"UA THICK S
HAKE"
90 WAIT 50
100 GOTO 20
```

Now, to explain the use of colour more fully, we are going to take a simple program, and gradually elaborate it, showing how adding such things as *user-defined graphics* can add a great deal of interest to your programs. At the end of this section, we'll give you a couple of suggestions to apply if you wish to keep improving the program, because, as every programmer knows, there is no such thing as a perfect program, there is always room for improvement.

The program we're going to use as the core of our development work is a standard 'Duck shoot' program, in which little objects fly across the screen, and you have to try and shoot them down. In the first version of the program, the little objects are letters, and you are the 'X'. You fire at the 'ducks' by pressing the space bar, and you move yourself left using the cursor keys, on either side of the space bar, (you move in the direction of the arrows on those keys).

Although there is no time limit on this program, there is a limit to the number of shots you fire. In all of the versions of this game in this part of the book, you'll see the program starts with a limit of 15 shots. The number of shots is deliberately kept low in the game, so you will not be able to get a high score just by leaving your finger on the space bar, and waiting for the ducks to fly into the line of fire.

Here, then is our first verion. Type it into your computer, and run it to get underway.

```
10 REM DUCK SHOOT
20 SC=0:SHOTS=15:CLS
30 A$="ZXC KD FI    TR SKF  AS  FK  FKKL
 D Q "
40 ACROSS=15:DOWN=14
50 PLOT 1,5,A$
60 PLOT ACROSS-1,DOWN," X "
70 IF DEEK(783)=48382 THEN GOSUB 150
80 PLOT 1,0,"SCORE "+STR$(SCORE)
90 PLOT 20,0,"SHOTS LEFT"+STR$(SHOTS)+"
 "
100 IFSHOTS<1THENPLOT7,10,"THAT'S THE  E
ND OF THE GAME":END
110 IF DEEK(783)=48351 THEN ACROSS=ACROS
S-1
```

```
120 IF DEEK(783)=48255 THEN ACROSS=ACROS
S+1
130 A$=RIGHT$(A$,37)LEFT$(A$,1)
140 GOTO 50
150 SHOTS=SHOTS-1
160 IF MID$(A$,ACROSS,1)=" " THENRETURN
170 SC=SC+57
180 L$=LEFT$(A$,ACROSS-1):R$=RIGHT$(A$,3
8-ACROSS)
190 A$=L$+" "+R$
200 RETURN
```

You'll see the letters which are held in A$ (see line 30) moving across near to the top of the screen. You (the "X") will be in about the middle of the screen when the program starts. You can — as we mentioned a few moments ago — move yourself back and forth using the cursor keys to get yourself into the position which you think gives you the best possible chance. When you judge a duck is directly overhead, press the space bar to fire your patented anti-duck missile. The number after the words 'SHOTS LEFT' (in the top right hand corner of the screen) will decrease, and if you have been accurate, the number after the word SCORE in the top left hand corner of the screen will increase.

Note, by the way, wherever possible, we have used *explicit names* for the variables within this program. That is, the variable name for the shots left is SHOTS, for your position across the screen, the variable name is ACROSS, and the position down the screen is DOWN. Even though it takes a little longer to type long variable names into a program and (of course) they use up more memory than do shorter names, running out of memory is rarely a problem on your computer, and the advantages of using explicit names to keep the purpose of various parts of the listing clear outweighs the extra time it takes to type them in.

If, for example, you were writing a program like this and then you decide that it would be better if the 'X' was printed further down the screen, you would not have to search through the program to work out which variable held your "down" co-ordinate. If you get into the habit of using explicit names, your debugging time will be that much shorter.

Line 30 defines the string variable A$ as a long series of letters and spaces. The letters can be anything you like; don't feel you have to copy ours. The important thing, however, is that the string is 38 characters long. You can check this by running the program briefly, stopping it with CTRL/C, then typing in, as a direct command, PRINT LEN(A$). If your string is the correct length, PRINT LEN(A$), followed by RETURN will give the answer 38.

The string handling, i.e. LEFT$, RIGHT$, MID$, etc., is used to move the ducks across the screen. The string A$ holds the ducks, as you will have gathered, and is plotted onto the screen. Then, using string slicing, a character is taken off one end, and placed at the other, printed again and the loop continues. So creating a feeling of movement.

If you are pressing the space bar when the computer gets to line 70 (and notice the use of the word 'DEEK', we'll cover that word later, it is extremely useful), then the action is sent to the subroutine at 150, where the value of shots is decreased by one. If there is a space at position MID$(A$,ACROSS,1) then the computer RETURNs from the subroutine because you have missed the ducks. If there is anything but a space there then the computer continues through the subroutine, adding 57 to your score, and removing the duck from the string.

Now all this takes some time to explain, but you'll find the computer does it apparently instantaneously. You press the space bar, the score increases by 57 (if you're a good shot), the number of shots drops by one and the duck disappears. You'll see that the program continues until when you run out of shots,

when the game terminates. Take note of your score at this point, and see if you can beat it in subsequent runs.

Once you have the program running to your satisfaction, and you have a pretty good idea of how it works, modify it to read like the following program. You do not have to NEW the computer. Just compare the program with the following listing, line by line, using the EDIT function where necessary and also adding new lines in between the old lines.

```
10 REM DUCK SHOOT
15 PAPER 0:INK 1
16 COL=1
20 SC=0:SHOTS=15:CLS
30 A$="ZXC KD FI    TR SKF  AS  FK  FKKL
 D Q "
40 ACROSS=15:DOWN=14
50 PLOT 1,5,A$
55 PLOT 0,5,COL
56 COL=COL+1:IF COL>7 THEN COL=1
60 PLOT ACROSS-1,DOWN," X "
70 IF DEEK(783)=48382 THEN GOSUB 150
80 PLOT 1,0,"SCORE "+STR$(SCORE)
90 PLOT 20,0,"SHOTS LEFT"+STR$(SHOTS)+"
"
100 IFSHOTS<1THENPLOT7,10,"THAT'S THE  E
ND OF THE GAME":END
110 IF DEEK(783)=48351 THEN ACROSS=ACROS
S-1
120 IF DEEK(783)=48255 THEN ACROSS=ACROS
S+1
130 A$=RIGHT$(A$,37)+LEFT$(A$,1)
140 GOTO 50
150 SHOTS=SHOTS-1
```

```
160 IF MID$(A$,ACROSS,1)=" " THENRETURN
170 SC=SC+57
180 L$=LEFT$(A$,ACROSS-1):R$=RIGHT$(A$,3
8-ACROSS)
190 A$=L$+" "+R$
200 RETURN
```

Now when you run this, you'll see an immediate and quite striking improvement. Colour certainly adds a lot to any program on your computer. Line 15 sets the PAPER (that is, the background) colour to black (colour number Ø) and the INK (the foreground colour) to red (INK colour 1).

Even if we had nothing else, there is a significant improvement in the program. You have the score, shots left, ducks and the 'X' all printed in red on a black background, which is far more interesting than just plain old black and white.

The other colour features like the 'shimmering' ducks are plotted onto the screen, like an escape character colour.

Apart from the colour changes we've discussed, the listing is essentially the same as the first listing. However, you can see that the few changes that we have made have improved the game considerably. We'll now continue with the improvements, adding such things as sound.

```
10 REM DUCK SHOOT
15 PAPER Ø:INK 1
16 COL=1:SHOOT:PLAY Ø,1,1,2ØØØ:WAIT 1ØØ
20 SC=Ø:SHOTS=15:CLS
30 A$="ZXC KD FI    TR SKF  AS  FK  FKKL
 D Q "
40 ACROSS=15:DOWN=14
50 PLOT 1,5,A$
```

```
55 PLOT 0,5,COL
56 COL=COL+1:IF COL>7 THEN COL=1
57 PLAY 1,0,1,1
60 PLOT ACROSS-1,DOWN," X "
70 IF DEEK(783)=48382 THEN GOSUB 150
80 PLOT 1,0,"SCORE "+STR$(SCORE)
90 PLOT 20,0,"SHOTS LEFT"+STR$(SHOTS)+"
"
100 IFSHOTS<1THEN GOTO 210
110 IF DEEK(783)=48351 THEN ACROSS=ACROS
S-1
120 IF DEEK(783)=48255 THEN ACROSS=ACROS
S+1
130 A$=RIGHT$(A$,37)+LEFT$(A$,1)
140 GOTO 50
150 SHOTS=SHOTS-1
160 IF MID$(A$,ACROSS,1)=" " THENRETURN
170 SC=SC+57
180 L$=LEFT$(A$,ACROSS-1):R$=RIGHT$(A$,3
8-ACROSS)
190 A$=L$+" "+R$
200 RETURN
210 PLOT7,10,"THAT'S THE  END OF THE GAM
E"
220 EXPLODE:END
```

There is very little difference between the second and third versions of Duck Shoot, apart from the PLAY and SHOOT commands, all is the same. As you know from previous chapters, the sound on the Oric adds a great deal of interest to the programs.

The next version of the game we'll look at is, as you can see,

considerably different from the ones we've been examining to date. (It's also a lot longer).

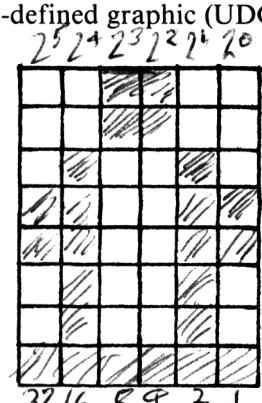
```
10 REM DUCK SHOOT
12 GOSUB 300
15 PAPER 0:INK 1
16 COL=1:SHOOT:PLAY 0,1,1,2000:WAIT 100
20 SC=0:SHOTS=15:CLS
30 A$="▯▯▯ ▯  ▯  ▯   ▯    ▯ ▯▯  ▯ ▯▯ ▯▯
▯▯    "
40 ACROSS=15:DOWN=14
50 PLOT 1,5,A$
55 PLOT 0,5,COL
56 COL=COL+1:IF COL>7 THEN COL=1
57 PLAY 1,0,1,1
60 PLOT ACROSS-1,DOWN," X "
70 IF DEEK(783)=48382 THEN GOSUB 150
80 PLOT 1,0,"SCORE "+STR$(SCORE)
90 PLOT 20,0,"SHOTS LEFT"+STR$(SHOTS)+"

100 IFSHOTS<1THEN GOTO 210
110 IF DEEK(783)=48351 THEN ACROSS=ACROS
S-1
120 IF DEEK(783)=48255 THEN ACROSS=ACROS
S+1
130 A$=RIGHT$(A$,37)+LEFT$(A$,1)
140 GOTO 50
150 SHOTS=SHOTS-1
160 IF MID$(A$,ACROSS,1)=" " THENRETURN
170 SC=SC+57
180 L$=LEFT$(A$,ACROSS-1):R$=RIGHT$(A$,3
8-ACROSS)
```

```
190 A$=L$+" "+R$
200 RETURN
210 PLOT7,10,"THAT'S THE END OF THE GAM
E"
220 EXPLODE:END
300 FOR A=46080+(8*ASC("!")) TO 46080 +(
8*ASC("!")+7)
310 READ US:POKE A,US
320 NEXT A:RETURN
330 DATA 0,19,54,30,14,3,0,0
```

You may be wondering why the letters in line 30 have been changed to exclamation marks (!), and if you have already run the program, you may be wondering why the exclamation marks are now little ducks. The answer is simple, the ducks have been user defined, the exclamation mark has been re-shaped as a duck.

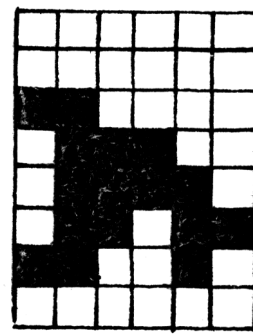
It's very simple to define a graphic. We'll take you through the way we created the alleged duck, and from this explanation you should be able to produce anything you like.

The basis of any user-defined graphic (UDG) is a six by eight grid, like this one:



To work out a graphic, you simply fill in the squares in this grid

which you wish to print as solid dots in the final graphic. Our duck (in grid form) looks like this:



If you look to the lines from 3ØØ onwards in the most recent listing of DUCK SHOOT, you'll see a loop, which includes a READ statement and the disturbing looking line, POKE A,US etc. This needs detailed explanation.

The shape of each character is held in the area of memory 46ØBØ to 48121Ø. You will probably only be concerned with the locations 46336 (the start of the space character, to 471Ø3 (the end of the 'hash' graphic). Each character takes 8 locations, each location holds a number for each row of the graphic. To find the start location for a particular graphic, i.e. "£", a short and easy way is to say, as a direct command;

```
PRINT 46ØBØ + (8*ASC("£"))
```

You will notice that the loop in line 3ØØ begins at the start location for the exclamation mark, and finishes 7 locations on, the end of the exclamation mark.

This short method of finding the start location for a particular character can be very useful when you want to define more than one graphic, as long as the characters you want to be redefined follow each other, (see the reference chart at the end of the book), then you could, if you wanted to redefine the "A" and

the "B", say 'FOR A = 46Ø8Ø + (8*(ASC("A")) TO 46Ø8Ø + (8*ASC("B")) + 7' followed by a similar routine as the one in subroutine 3ØØ.

The idea of locations, PEEK, POKE, DEEK and DOKE will be covered in the last chapter of the book.

Now we have learned where to put the information about our new characters, we now need to know what to put there.

As we have mentioned before, each character takes up eight locations for the eight rows of information, and in each location there is a number between Ø–63 which holds the information for six columns across (it actually holds information for eight columns, but we only use six of them), stored in binary.

A binary number is a weird looking thing, it's all Ø's and 1's, nothing else. This is very useful as far as our graphic is concerned, as when writing down the binary number that we want to poke into a particular location, we can just skim across a row and, if a square is empty, we write a zero and if it is full then we write a 1. Our duck looks something like this in binary form:

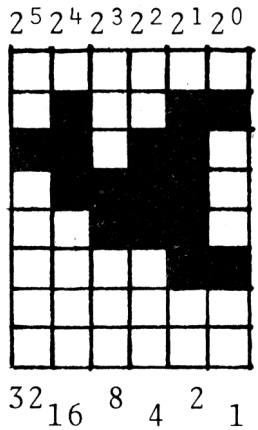
ROW 1 ØØØØØØ
ROW 2 Ø1ØØ11
ROW 3 11Ø11Ø
ROW 4 Ø11111
ROW 5 ØØ1111
ROW 6 ØØØØ11
ROW 7 ØØØØØØ
ROW 8 ØØØØØØ

Can you see the pattern?

You may already have spotted the problem about 'how does the Oric know when you're talking about binary rather than

decimal?' Well it can't. You have to convert between binary and decimal before you can 'POKE' into the required location.

Converting between decimal and binary is not an easy thing for the beginner, but it is a useful thing to learn. Look again, at our duck. Study this diagram carefully, as it holds a lot of useful information:



$=0$

$2^4+2^1+2^0=19$

$2^5+2^4+2^2+2^1=54$

$2^4+2^3+2^2+2^1=30$

$2^3+2^2+2^1=14$

$2^1+2^0=19$

$=0$

$=0$

The only thing that you really need to remember is the numbers along the bottom of the grid. As you work your way along each row, if a block is set, that is filled in, then add the corresponding column number, i.e. in row 2 of the duck the decimal number is 16 + 2 + 1 because those columns are set.

We can understand that this might be a little confusing, so here is a program which will do the conversion for you:

```
10 REM BINARY TO DECIMAL CONVERTER
20 REM PETER SHAW 1983
30 INPUT "BINARY ";B$:T=0
40 IF LEN(B$)<>6 THEN 30
50 FOR A=0 TO 5
```

```
60 IF MID$(B$,6-A,1)="1" THEN T=T+(2^A)
70 NEXT A
80 PRINT "DECIMAL ";T:PRINT
90 GOTO 30
```

That brings us to the end of this series of DUCK SHOOT games. There are five things you can do to further develop the program:

- Cut the number of shots available to make the program even more challenging
- User define the "X", it would be best to include it as a separate subroutine which is called right at the start of the game.
- Add a 'high score' feature!
- Allow the computer to detect when all the ducks have been shot (this will happen when A$ is just full of 38 spaces) and when they have all been shot, to give a bonus, more shots and a new row of duck to fire at.
- Add an extra line of ducks (B$) flying in the opposite direction.

CHAPTER THIRTEEN
GETTING LISTED

An *array* is used when you want to create a list of items, and refer to the item by mentioning just which position in the list it occupies. You set up an array by using the DIM command. If you type in DIM A(5), the computer will set up a list in its memory called A, and will save space for five items: A(1), A(2), A(3), A(4) and A(5). These, by the way, are called *elements* of the array.

When you *dimension*, or set up, an array, the computer creates the list in its memory, and then fills every item in that list with a zero. So, if you said to the computer, PRINT A(1) or PRINT A(4) it would come back with Ø. You will fill items in an array with LET (such as LET A(2) = 1ØØØ) or using READ and DATA as we saw in Chapter eleven. Once you've filled a position in the list, whether you filled it with LET or READ, you can get the computer to give you the contents of that element of the array, by simply saying PRINT A(2).

The next program *dimensions* (this, as we said, simply means sets up, or creates) an array called A, with room for fifteen elements. The B loop from lines 3Ø to 5Ø fills the array with random digits between zero and nine, and then prints them back for you with the loop from lines 6Ø to 9Ø.

```
10 REM ARRAYS
20 DIM A(15)
30 FOR B=1 TO 15
40 A(B)=INT(RND(1)*9)
50 NEXT B
60 FOR Z=1 TO 15
70 PRINT "A("Z") IS "A(Z)
80 WAIT 10
90 NEXT Z
```

This is called a *one-dimensional* array, because a *single digit* follows the letter which "labels" the array. You can also have *multi-dimensional* arrays, in which *more than one number* follows the array label after DIM. In the next program, for example, the computer sets up a two-dimensional array called A, consisting of four elements (that is, it is dimensioned by DIM A(4,4) as you can see in line 20:

```
10 REM MULTI-D ARRAYS
15 INK 7:PAPER0
16 CLS
20 DIM A(4,4)
30 OR B=1 TO 4
40 FOR C=1 TO 4
50 A(B,C)=INT(RND(1)*9)
60 NEXT C,B
70 PRINT CHR$(27);"T    1 2 3 4"
80 FOR B=1 TO 4
90 PRINT CHR$(27);"T"B;CHR$(27)"P";
100 FOR C=1 TO 4
110 PRINT A(B,C);
120 NEXT C
130 PRINT
140 NEXT B
```

When you run it, you will see something like this:

```
   1 2 3 4
1  7 8 8 0
2  8 7 8 7
3  6 3 2 2
4  3 5 8 7
```

You specify the element of a two-dimensional array by referring to *both* its numbers, so the first element of this array (the number 7 in the top left hand corner of the printout above) can be referred to as A(1,1). The 0 at the end of the line is A(1,4) and the 8 on the bottom row is A(4,3).

Your computer also supports *string* arrays. Enter and run this routine to see a string array in operation:

```
10 REM STRING ARRAYS
20 DIM A$(5)
30 FOR B=1 TO 5
40 A$(B)=CHR$(INT(RND(1)*26)+65)
50 NEXT B
60 FOR B=1 TO 5
70 PRINT "A$("B") IS "A$(B)
80 NEXT B
```

You need not only put one letter into each element, you can, if you wish, use much longer strings, as in this example:

```
10 REM STRING ARRAYS
20 DIM A$(5)
30 FOR A=1 TO 5
40 READ A$(A)
50 NEXT A
60 FOR A=1 TO 5
70 PRINT "A$("A") IS '"A$(A)"'"
80 NEXT A
```

```
90 DATA ORIC,ONE,STRING,ARRAYS,ARE DEMON
STRATED HERE
```

ESCAPING FROM MURKY MARSH

This program demonstrates the use of a two-dimensional array for "holding" an object and for printing it out. The object in this case is a miniature dragon, who is trying to escape from Murky Marsh, indicated in this program by a square of dots. Our dragon is very stupid, and moves totally at random within the marsh. He is free if he manages to stumble onto the outer right or bottom two rows.

The dragon in this program demonstrates *Brownian* motion, the random movement shown by such things as tiny particles in a drop of water viewed under a microscope, or of a single atom in a closed container. Brownian motion explains why a drop of ink gradually mixes in with the water into which it has been placed.

Here is the program listing:

```
10 REM DRAGON'S LAIR
20 DIM A(10,10):M=0
30 GOSUB 500
40 X=INT(RND(1)*2)
50 IF X=0 THEN P=P+1
60 IF X=1 THEN P=P-1
70 X=INT(RND(1)*2)
80 IF X=0 THEN Q=Q+1
90 IF X=1 THEN Q=Q-1
100 IF Q<1 THEN Q=Q+1
110 IF P<1 THEN P=P+1
120 M=M+1
121 CLS
130 PLOT 3,3,"Attempt "+STR$(M)+" "
```

```
410 A(P,Q)=-13
415 PRINT:PRINT:PRINT:PRINT:PRINTTAB(17)
;
420 FOR X=1 TO 10
430 FOR Y=1 TO 10
440 PRINT CHR$(A(X,Y)+46);
450 NEXT Y
460 PRINT:PRINT TAB(17);
470 PLAY 0,1,1,10:NEXT X
480 A(P,Q)=0
490 IF Q>8 OR P>8 THEN GOTO 600
495 GOTO 40
500 Q=INT (RND(1)*3)+4:P=INT (RND(1)*3)+
4
510 FOR B=46080+(8*33) TO 46080+(8*33)+7
520 READ US
530 POKE B,US
540 NEXT B
550 PAPER 0
560 INK 6:CLS
570 RETURN
580 DATA 0,16,58,31,46,7,6,30
600 PRINT:PRINT:PRINT:PRINT:PRINT
610 PRINT "Whew...Free at last",
```

DAVY JONES' LOCKER

You can now relax with our next game — FULL FATHOM FIFTY — which uses two arrays (B and D) to store the player's scores and dice rolls respectively. It is a simple game. You and the computer are in a race to roll a total of 50 or more. You roll two dice at once, but the only time you score and get the total of

the dice roll added to your accumulating total, is when both dice come up with the same number.

You should know enough about how programs work at this stage to be able to determine what each section does, so we will not explain this relatively simple program. Get it up and running and decide for yourself how each section works. You'll possibly gain more from the program by working it out yourself rather than having it explained in detail by us.

Here is the program listing for Full Fathom Fifty:

```
10 REM FULL FATHOM FIFTY
15 PAPER 0:CLS:INK 2
20 DIM B(2):DIM D(2):DIM A$(38):ROUND=1
30 FOR A=1 TO 2
40 PLOT 8,5,"Full Fathom Fifty"
50 PLOT 9,24,"Round number "+STR$(ROUND)
+" "
55 PLOT 13,10,"ORIC: "+STR$(B(1))
56 PLOT 12,12,"HUMAN: "+STR$(B(2))
60 IF B(1)>49 OR B(2)>49 THEN 230
70 IF A=1 THEN PLOT 9,14,"MY TURN TO ROL
L !"
80 IF A=2 THEN PLOT 8,14,"PRESS 'R' TO R
OLL":GET A$
90 ZAP:ZAP:ZAP
100 PLOT 1,14,"
                "
110 FOR C=1 TO 2
120 PLOT 8,14,"ROLLING DIE No."+STR$(C)
130 FOR E=1 TO 10:EXPLODE:WAIT A*5:NEXT
E
140 D(C)=INT(RND(1)*6)+1
```

```
150 PLOT 5,14,"          IT CAME UP "+STR$
(C))+"          "
160 PING:WAIT 100
165 NEXT C
166 WAIT 100
170 IF D(1)<>D(2) THEN PLOT 8,14,"NO DOU
BLE-NO SCORE"
180 IFD(1)=D(2)THENPLOT8,14,"DOUBLE!! SC
ORES "+STR$(3*D(1)):B(A)=B(
A)+3*D(1)
185 WAIT 150
190 SHOOT:PLOT 1,14,"
                    "
210 NEXT A
220 ROUND=ROUND+1:GOTO 30
230 PLOT 3,24,"And the winner is ....."
235 WAIT50
240 IF B(1)<B(2) THEN PLOT 28,24,"YOU !!
"
250 IF B(1)>B(2) THEN PLOT 28,24,"ME !!"
260 FOR A=1 TO 10:EXPLODE:WAIT A*10:NEXT
```

Here's how the program can look:

```
        Full Fathom Fifty

            ORIC: 3

           HUMAN: 47
```

```
PRESS 'R' TO ROLL

  Round No. 8
```

CHAPTER FOURTEEN
LET THE GOOD GRAPHICS ROLL

Your computer has a very wide range of graphic abilities, and in this chapter, we'll introduce you to those abilities. After the introduction of each function, we'll give you a program to demonstrate its use.

These demonstrations should be seen only as *introductions* to the graphics potential of your computer. Experimentation will demonstrate just how far you can stretch the Oric.

PLOT LX,LY,CN

Plot, as you will have gathered from previous chapters, can be used to put information at a specific point on the screen. But plot can also be used to put dots or colour onto the screen. Before you can plot onto the screen in single blocks, it is best to put the computer into the correct LORES mode. For normal operation, and normal plotting this should be LORES Ø, but when you want to use the alternate character set, then you need to use LORES 1. The numbers which follow the PLOT command are:

LX — the X co-ordinate for the low-res. screen Ø–38
LY — the Y co-ordinate for the low-res. screen Ø–26
CN — the ASCII value of the character, or control character to be printed. PLOT can be used in the same way as an escape character.

Run the next program, entering numbers between zero and 38 for the X value, and zero and 26 for the Y value. This program plots CHR$(17), (the control character for a red background), onto the co-ordinates X,Y:

```
10 REM PLOT
20 INPUT "ENTER X    ";X
30 INPUT "ENTER Y    ";Y
40 CLS:LORES 0
50 PLOT X,Y,17
60 GOTO 20
```

You can get the computer to generate PLOT co-ordinates at random, and then 'reflect' them into each of the four corners of the screen as our next program convincingly demonstrates:

```
10 REM BUTTERFLY PLOT
15 LORES 0
20 X=INT(RND(1)*19)+1
30 Y=INT(RND(1)*13)
40 PLOT X,Y,18
50 PLOT 38-X,Y,18
60 PLOT X,26-Y,18
70 PLOT 38-X,26-Y,18
80 GOTO 20
```

Add colour, and see how extraordinary the result can be:

```
10 REM BUTTERFLY PLOT Mk II
12 REM PRESS ANY KEY TO CLEAR SCREEN
15 LORES 0
20 X=INT(RND(1)*19)+1
25 IF KEY$<>"" THEN LORES 0
30 Y=INT(RND(1)*13)
35 C=INT(RND(1)*7)+17
```

```
40 PLOT X,Y,C
50 PLOT 38-X,Y,C
60 PLOT X,26-Y,C
70 PLOT 38-X,26-Y,C
80 GOTO 20
```

A HIGHER GRAPHIC ABILITY

You may have thought that the PLOT blocks were a little large to be any real use in graphic programs. Most of the time you will find that they are much too large. Your Oric is equipped with a command which will change your whole outlook on life, HIRES, which stands for High-Resolution. When the computer is given this command it does a very weird thing, try it and see. Type HIRES followed by RETURN. Did you see that flash? That was the computer changing from the crude, 38 by 26 resolution display to the much finer, and much more fun, 239 by 199 display.

FINER PLOTTING

In this HIRES mode, you cannot use PLOT, you need another command. Of course, this command is available, but can be a little confusing at first. CURSET HX,HY,FB;

HX — the X co-ordinate for the Hi-res. screen Ø–239
HY — the Y co-ordinate for the Hi-res. screen Ø–199
FB — the Foreground/Background number (as follows)

Ø	In present background INK
1	In present foreground INK
2	Invertion between present background and foregrou INK's
3	Null, has no effect on display.

When you 'curset' a point onto a high resolution screen you will notice an immediate difference, how small the point is. Before you can fully realise the potential of the CURSET command, you will have to try these short programs which 'curset' a series

of mathematical curves onto the screen, using the functions PI, (π), SIN, (sine) and COS (cosine) to create a wonderful display.

```
10 REM SIN CURSET
20 PAPER 0:HIRES:INK 6
30 FOR A=0 TO 2*PI STEP .01
40 CURSET33*A+17,100+100*SIN(A),1
50 NEXT A

10 REM COS CURSET
20 PAPER 0:HIRES:INK 6
30 FOR A=0 TO 2*PI STEP .01
40 CURSET33*A+17,100+100*COS(A),1
50 NEXT A

10 REM SIN/COS CURSET
20 PAPER 0:HIRES:INK 6
30 FOR A=0 TO 2*PI STEP .009
40 CURSET33*A+17,100+100*COS(A),1
50 CURSET33*A+17,100+100*SIN(A),1
60 NEXT A

10 REM STARFLIGHT
20 PAPER 0:HIRES:INK 5
30 FOR B=10 TO160 STEP 20
40 FOR A=0 TO 2*PI STEP .01
50 CURSET 10*A+B,80*COS(A)+110,1
60 CURSET 10*A+B,80*SIN(A)+110,1
70 NEXT A,B

10 REM OVAL
20 PAPER 0:HIRES:INK 5
```

```
30 FOR A=1 TO 105 STEP .3
40 B=PI*A/50:C=95*COS(B)+125
50 D=79*SIN(B)+83
60 CURSET C,D,1
70 CURSET C/2,D,1
80 CURSET C,D/2,1
90 CURSET C/2,D/2,1
100 NEXT A
```

DRAW RX,RY,FB

Draw is a very useful command on the Oric, it allows you to draw a line from the last point you 'curseted' or drew to, to a point on the screen RX,RY plus the current cursor co-ordinates.

RX — Relative draw X co-ordinate
RY — Relative draw Y co-ordinate
FB — As for CURSET.

In our first DRAW program, the computer 'cursets' an X,Y position (chosen at random by the subroutine from line 70), then draws three lines from this, 'curseting' X,Y after each line is drawn to 'restart' the new line from the same position as the other lines have used for a beginning point.

```
10 REM DRAW
15 PAPER 0:HIRES:INK 6
20 GOSUB 70
30 CURSET X,Y,1:DRAW X,199-Y,1:WAIT 20
40 CURSET X,Y,1:DRAW 239-X,Y,1:WAIT 20
50 CURSET X,Y,1:DRAW 239-X,199-Y,1:WAIT
20
60 GOTO 20
70 X=RND(1)*117
80 Y=RND(1)*99
```

```
90 WAIT 10
100 RETURN
```

Our second draw program adds a little colour:

```
10 REM DRAW II
15 PAPER 0:HIRES:INK 6
20 GOSUB 70
30 CURSET X,Y,1:DRAW X,199-Y,1:WAIT 20
40 CURSET X,Y,1:DRAW 239-X,Y,1:WAIT 20
50 CURSET X,Y,1:DRAW 239-X,199-Y,1:WAIT
20
60 GOTO 20
70 X=RND(1)*117
80 Y=RND(1)*99
90 WAIT 10
95 INK RND(1)*7+1
100 RETURN
```

This can be further developed to produce the dynamic program **WARP SPEED:**

```
10 REM WARP SPEED
20 PAPER 0:HIRES:INK 6
30 X=RND(1)*117
40 Y=RND(1)*99
50 CURSET 117,99,0
60 DRAW X,Y,1
70 CURSET 117,99,0
80 DRAW-X,Y,1
90 CURSET 117,99,0
100 DRAW X,-Y,1
110 CURSET 117,99,0
120 DRAW-X,-Y,1
```

```
130 PATTERN RND(1)*255
140 GOTO 30
```

In Warp Speed, a point is 'curseted' in the middle of the screen, and then four lines are drawn out in a symmetrical form. Line 130 holds a command we haven't seen before, PATTERN. This new command changes the build up of the drawn line. You remember how we create one row of binary numbers to represent blocks in a row of a UDG? well the DRAW line is built up in a similar way, only using a row with eight columns instead of 6. What line 130 is doing is choosing a random number to make the drawn line less 'solid'.

Our next program, based on one written by Jeremy Ruston, uses CURSET and DRAW to remarkably good effect:

```
10 REM     Urban Pacifier
20 REM Based on 'String Art'
30 REM    by Jeremy Ruston
40 PAPER 0:HIRES:INK 6
50 X=INT(RND(1)*239)
60 Y=INT(RND(1)*199)
70 L=INT(RND(1)*239)
80 M=INT(RND(1)*199)
90 U=15:V=7
100 DEF FN R(X)=INT (RND(1)*X)
110 GOSUB 270
120 NUM=NUM-1
130 IF NUM=0 THEN GOSUB 270
140 CURSET X,Y,1
150 DRAW L-X,M-Y,1
160 PLAY 1,0,1,10
170 IF X+A>238 OR X+A<1 THEN A=-A
180 IF Y+B>198 OR Y+B<1 THEN B=-B
190 IF L+C>238 OR L+C<1 THEN C=-C
```

```
200 IF M+D>198 OR M+D<1 THEN D=-D
210 PLAY 0,1,1,10
220 X=X+A:Y=Y+B
230 L=L+C:M=M+D
240 CO=FN R(200)
250 IF CO<4 THEN RUN
260 GOTO 120
270 A=FN R(U)-U
280 B=FN R(U)-U
290 C=FN R(U)-U
300 D=FN R(U)-U
310 NUM=FN R(20)+10
320 INK RND(1)*7+1
330 U=RND(1)*6+7
340 RETURN
```

CIRCLE R,FB

Not only can your wonderful machine draw lines here, there and everywhere, it can also draw circles! Circle is quite simple to use. First of all you 'curset' the center of the circle where you want it on the screen, then you tell the computer to draw a CIRCLE radius R, Foreground/Background colour (see CURSET).

The next two programs both use circle, the first draws them randomly about the screen, while the second also employs the command PATTERN again to produce a dotted circle outline.

```
10 REM CIRCLING AROUND
20 PAPER 0:HIRES:INK 6
30 X=INT(RND(1)*179)+30
40 Y=INT(RND(1)*138)+30
50 R=INT(RND(1)*30)+1
60 CURSET X,Y,0
```

```
70 CIRCLE R,1
75 IF RND(1)>.97 THEN HIRES
80 GOTO30

10 REM CIRCLING AROUND WITH PATTERN
20 PAPER 0:HIRES:INK 6
30 X=INT(RND(1)*179)+30
40 Y=INT(RND(1)*138)+30
50 R=INT(RND(1)*30)+1
60 CURSET X,Y,0
65 PATTERN RND(1)*255
70 CIRCLE R,1
75 IF RND(1)>.97 THEN HIRES
80 GOTO30
```

MULTI-COLOURS

You are generally restricted to eight colours on your Oric, (counting black and white as colours). However, with the use of user defined graphics, and the plot command, we can easily mix a few colours to produce some attractive hues.

```
10 REM RING THE CHANGES
20 PAPER 0:INK 6:CLS
30 POKE 46344,21
40 POKE 46345,42
50 POKE 46346,21
60 POKE 46347,42
70 POKE 46348,21
80 POKE 46348,42
90 POKE 46349,21
100 POKE 46350,42
110 A$="Q"
```

```
120 FOR T=1 TO 6:A$=A$+A$:NEXT T
130 A$=LEFT$(A$,33)
140 FOR A=17 TO 23
150 FOR B=0 TO 7
155 PLOT 8,4,"BACKGROUND "+STR$(A-17)
160 PLOT 0,B+10,A:PLOT 1,B+10,B:PLOT 2,B
+10,A$
165 PLOT 36,B+10,16:PLOT 37,B+10,STR$(B)
170 NEXT B
180 PLOT 10,26,"PRESS A KEY":GET A$
190 NEXT A
200 GOTO 140
```

DO NOT ADJUST YOUR SET

This program by Paul Holmes demands two user-defined graphics to create a rather pretty display, useful if you want to tune your T.V. into the best picture.

```
10 REM DO NOT ADJUST YOUR SET
20 REM Paul Holmes
30 RESTORE
40 FOR A=46080+(8*ASC("#")) TO 46080+(8*
ASC("$")+7)
50 READ US:POKE A,US
55 NEXT A
56 I=0
60 DATA 7,7,7,7,7,7,7,7,62,62,62,62,62,6
2,62,62
70 PAPER 0:INK 7:CLS
75 B$=CHR$(126)+CHR$(126)+"####"+CHR$(12
6)+CHR$(126)
80 FOR Z=1 TO 6
```

```
90 FOR J=1 TO 3
100 PRINT TAB(22)CHR$(27);CHR$(Z+80);"$$
$$";
110 PRINT CHR$(27);CHR$(Z+81);B$;CHR$(27
);CHR$(Z+80);
120 PRINT"$$$$"CHR$(27);CHR$(80)
130 NEXT J,Z
140 PLOT 3,25,"PRESS A KEY FOR NEXT INK
COLOUR"
150 GET A$
160 INK I
165 PLOT 0,25,7:PLOT 1,25,16
170 I=I+1:IF I=8 THEN I=0
180 GOTO 140
```

OTHER HIRES COMMANDS

Less used, but still available are the commands like CURMOV, CHAR and FILL. CURMOV X,Y,FB is just like CURSET apart from the fact that X,Y are relative to the last point set. CHAR X,S,FB draws a normal, or alternate character on the HIRES screen at the last point set. S is the character set, (Ø for standard, i.e. LORES Ø, and 1 for alternate, i.e. LORES 1), X is the ASCII code for the letter or character to be printed, and FB is normal Foreground/Background.

FILL R,CC,N this command 'paints' in an area of the screen with a particular colour (as specified by the N, i.e. as in PLOTting colour). Note, when you wish to return back to ordinary character, low res. display, type TEXT followed by enter.

To get the most out of your graphics it is best if you experiment, apart from it being highly productive and educational, it also gives you some satisfaction to write some programs, rather than typing in ours all the time.

CHAPTER FIFTEEN
GETTING TAPED

You may have already realised that when you turn off your computer after a day working on it, you lose what you had typed in so laboriously. This need no longer be a problem, all you need is a blank cassette tape and a tape recorder which has a five-pin DIN socket. You should find amongst the goodies which came with your Oric, a three-pin DIN to a three-pin DIN connector, take this out and plug one end into your cassette recorder. The other end should be plugged into the right hand socket, (looking from the back), of your Oric. We are now ready to learn about saving programs to tape.

CSAVE

Put your blank cassette tape into the recorder, then type in this short program:

```
10 REM CSAVE TEST
20 PRINT "THIS IS A CSAVE TEST"
30 REM
40 REM
```

Type, as a direct command:

```
CSAVE "TEST"
```

DO NOT PRESS RETURN YET!

Press Record and Play on your cassette recorder, and wait for the leader to be wound into the tape so that you are recording

on the tape, not the clear plastic at the start. Now you can press RETURN.

After a short pause, you should see the message "SAVING ... TEST" appear in the top left hand corner of the screen, this tells you that your program is now being converted into sound which is being recorded onto your blank tape.

When the saving is complete, the message 'Ready' should appear.

That's all very well, but how do you get that program back into the computer?

CLOAD

Rewind your tape, type NEW followed by RETURN on your computer and LIST the program, just to check there is nothing left in memory.

Type, as a direct command:

```
CLOAD "TEST"
```

The message 'Searching ...' will appear in the top left again. Press play on your recorder, and set the volume to about three quarters. After a few seconds, with any luck, you should see the message 'Loading ... TEST'. Once it has loaded, list the program and check that there are no errors in the listing, if there are rewind the tape, choose a different volume setting and re-cload the program. When you make a successful CLOAD, mark the volume on your tape recorder so that you don't have to fiddle around every time you want to load a program. If you cannot get the tape to load after many tries, then there is a much more reliable, but a lot slower way of saving and loading. Get back to the original TEST program, and set the recorder and computer up as if you were going to make a normal CSAVE. Type as a direct command:

```
CSAVE "TEST",S
```

And carry out the necessary actions, i.e. record/play on the tape, etc.

To load a program which has been 'super saved', (i.e. saved at a slower, but much more reliable rate), type;

```
CLOAD "TEST",S
```

MORE TRICKS WITH LOAD AND SAVE

If, when you save at the normal rate, you include the word AUTO after the quotes, i.e.;

```
CSAVE "TEST",AUTO
```

then when you load the program, using;

```
CLOAD "TEST"
```

it will run automatically. Try it with the TEST program.

You can also save areas of memory. The screen is an area of memory called the display file. The display file for a low-res. screen is found from 48000 to 49119, you can save this area of memory by typing;

```
CSAVE "SCREEN",A48000,E49119
```

You can, in fact, save any area of memory you wish. To re-load the SCREEN, type;

```
CLOAD"SCREEN",A48000
```

This ends this short chapter on saving and loading. You can now prepare for the next chapter armed with as many blank tapes as you can get hold of. It contains many games, all worth a good save.

CHAPTER SIXTEEN
MORE GREAT GAMES

In this chapter, we'll be giving you the listing of a number of major programs. The notes on these will not be as extensive as you've had before with some of the programs in previous chapters, but the introductions to the programs will highlight some of the more interesting programming techniques used. Examination of the listings will give you a number of ideas you can apply to your own programs.

BAGATELLE

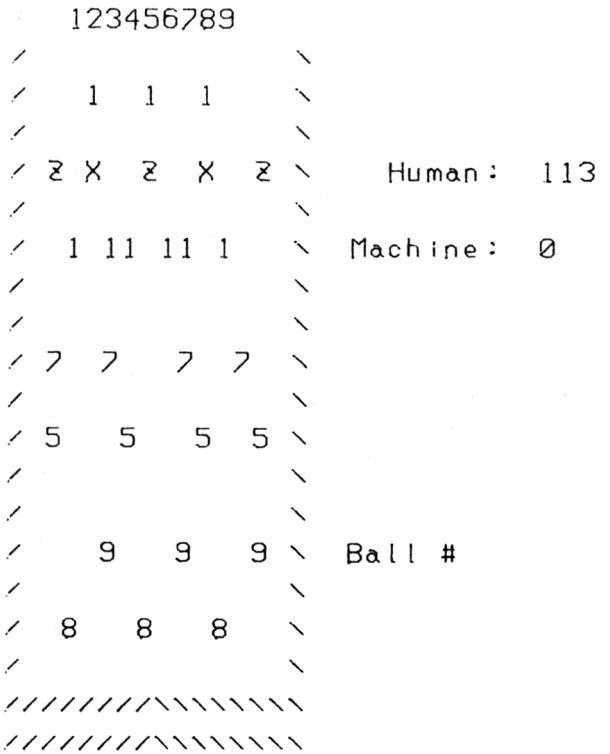
This is a simplified pinball machine, in which you and the computer take it in turn to roll diamond shaped balls down a frame which contains a series of numbers and letters. Your score tends to increase each time you hit an obstacle on the way down the frame, and the ball will bounce off the obstacles, increasing your chances of hitting even more of them. Hitting the sides of the frame, or bouncing around on the bottom, can cost you small penalties. In general though, each time the ball hits something your score will increase.

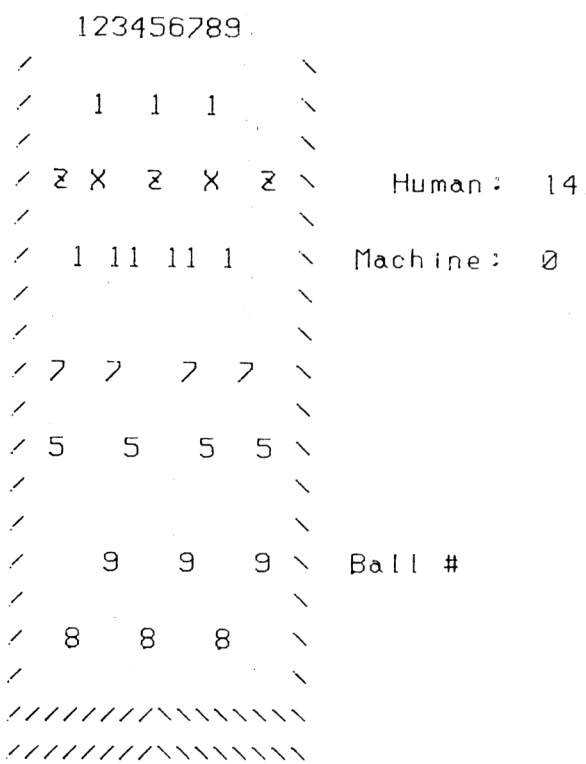
If you look at the sample printouts which follow this introduction, you'll see the numbers 1 to 9 along the top of the play area.

You start your ball rolling down the frame from any one of those. You and the computer have five balls each, to roll down the Bagatelle frame, and you take it in turns to do so. You have

the first roll. Then the computer will tell you which ball it intends to roll down the frame, and then roll this ball.

Here are some sample printouts;





Here's the listing to Bagatelle:

```
10 REM BAGATELLE
20 GOSUB 1000
30 GOSUB 500
40 FOR Y=1 TO 5:PLOT 25,14,STR$(Y)
50 FOR X=1 TO 2
60 IF X=2 THEN PLOT 25,14,STR$(Y)+" Me"
61 IF X=2 THEN K=INT(RND(1)*9)+1:PLOT 1,
21,"I will start with "
```

```
62 IF X=2 THEN PLOT 19,21,STR$(K):ZAP:WA
IT 100
63 IF X=2 THEN PLOT 1,21,"
                         "

70 IF X=1 THEN PLOT 28,14,"You"
71 IF X=1 THEN INPUT "Which No. to start
 with ";K
72 IF X=1 THEN IF K<1 OR K>9 THEN 71
80 PLOT 2+K,0,STR$(K):PLAY 0,1,1,20
85 PLOT 4,0,"123456789":PLOT 0,0,20
90 BA=K+3:EA=BA
100 BD=1:ED=1
110 IA=1
120 ID=1
130 PLOT EA,ED," ":EA=BA:ED=BD
150 PLOT BA,BD,"O"
160 FL=0:SA=SCRN(BA,BD+1)
170 IF SA<>32 THEN C(X)=C(X)+SA-48:FL=1:
GOTO 220
180 SA=SCRN(BA+1,BD)
190 IF SA<>32 THEN C(X)=C(X)+SA-48:FL=1:
GOTO 220
200 SA=SCRN(BA-1,BD)
210 IF SA<>32 THEN C(X)=C(X)+SA-48:FL=1
220 IF FLAG=1THEN PLAY1,1,1,20:IFRND(1)>
.3THENIA=-IA:ID=-ID
225 PLOT 28,4,STR$(C1)):PLOT 28,6,STR$(C
(2))
230 BD=BD+(ID*RND(1))+0.75
235 IF BA<3 THEN IA=-IA:BA=5
236 IF BD<1 THEN ID=-ID:BD=3
237 IF BA>15 THEN IA=-IA:BA=13
```

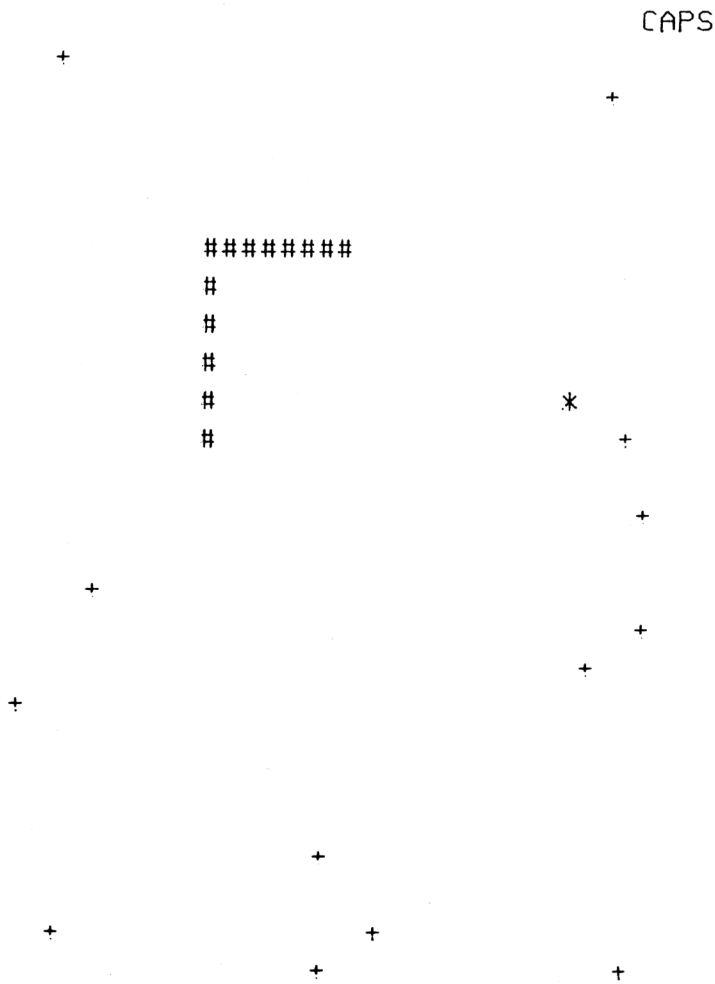
```
240 BA=BA+(IA*RND(1))
250 IF BD<19 THEN 130
260 PLAY 1,0,2,10
270 GOSUB 500
280 NEXT X,Y
300 PLOT 5,21,"The winner is"
310 IF C(1)<C(2) THEN PLOT 19,21,"the ma
chine"
320 IF C(2)<C(1) THEN PLOT 19,21,"the hu
man  "
330 IF C(2)=C(1) THEN PLOT 19,21,"-- A D
RAW --"
490 EXPLODE:END
500 CLS
501 PLOT 4,0,"123456789":PLOT 21,4,"Huma
n: "+STR$(C(1))
502 PLOT 0,0,20
505 PLOT 19,6,"Machine: "+STR$(C(2))
506 PLOT 19,14,"Ball #"
510 FOR Q=1 TO 19
520 PLOT 1,Q,"/":PLOT 16,Q,"\"
525 PLOT 0,Q,17:PLOT 18,Q,23:PLOT 17,Q,2
526 PRINT
530 NEXT Q
535 PLOT 1,18,"////////\\\\\\\\"
536 PLOT 0,18,17:PLOT 0,19,17
537 PRINT:PRINT:PRINT:PRINT:PRINT
538 PLOT18,18,23:PLOT18,19,23:PLOT17,18,
2:PLOT17,19,2
540 PLOT 1,19,"////////\\\\\\\\"
550 PLOT 5,2,"1  1  1":PLOT 3,4,"Z X  Z
```

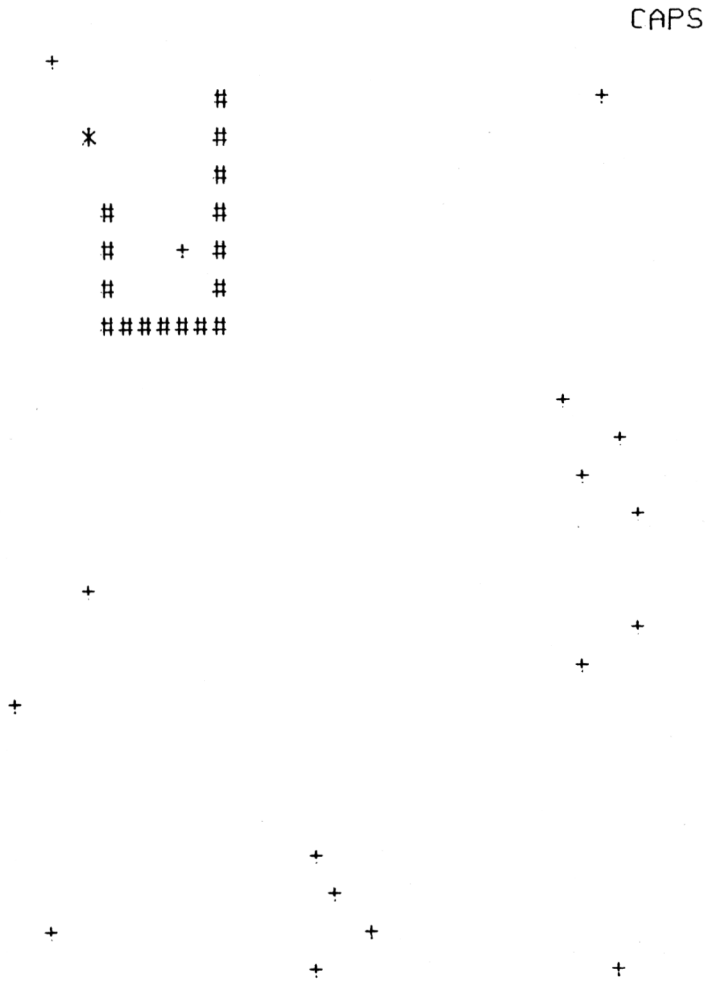
```
 X  Z"
560 PLOT 4,6,"1  11  11  1"
570 PLOT 3,9,"7   7    7   7"
580 PLOT 3,11,"5    5    5   5"
590 PLOT 6,14,"9    9    9":PLOT 4,16,"8
 8    8"
600 RETURN
1000 FOR A=46080+(8*(ASC("Q")))TO 46080+
(8*(ASC("Q"))+7)
1010 READ B
1020 POKE A,B
1030 NEXT A
1040 DATA 12,18,33,45,45,33,18,12
1050 DIMC(2)
1060 PAPER 0:INK 6:CLS
1070 RETURN
```

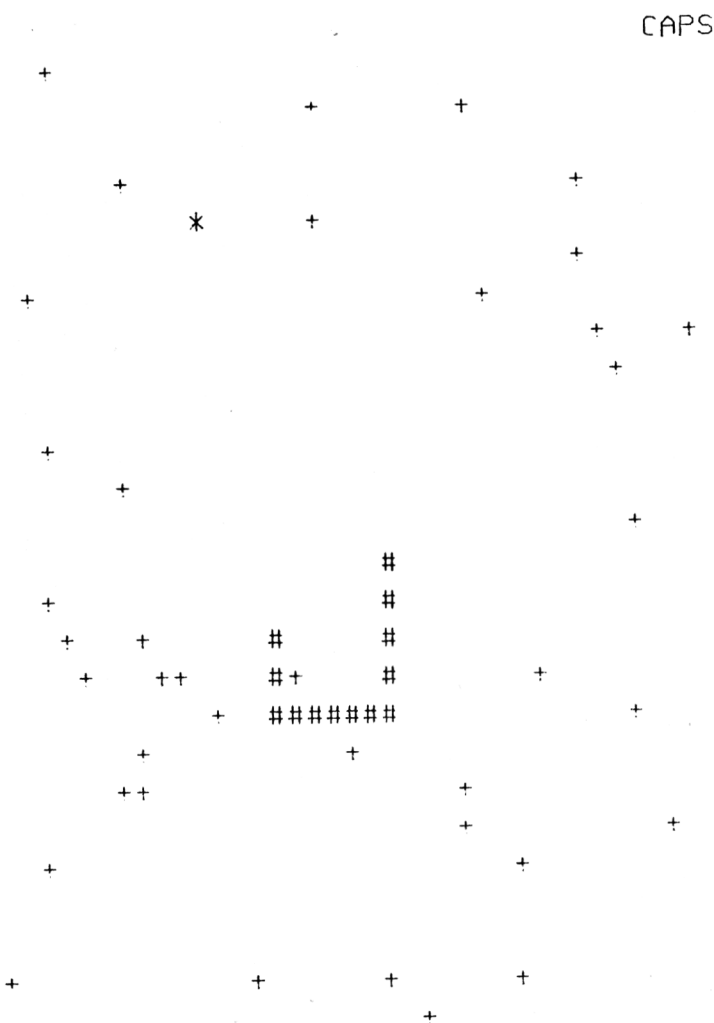
HISSING SID

In this program, by Paul Toland, you are in control of a snake, called Sid, and you must try to grow your snake as long as possible, by directing it to the asterisk signs (*) which it eats. However, the *'s remain on the screen for a decreasingly short time, after which they turn into poisonous plus signs (+). Poor Sid will die if he eats a + sign, or if he runs into the border or himself.

Once the game is over, you will be told how long Sid managed to become.
Here are a few sample printouts for Hissing Sid;


CAPS


CAPS


CAPS

Here's the program listing:

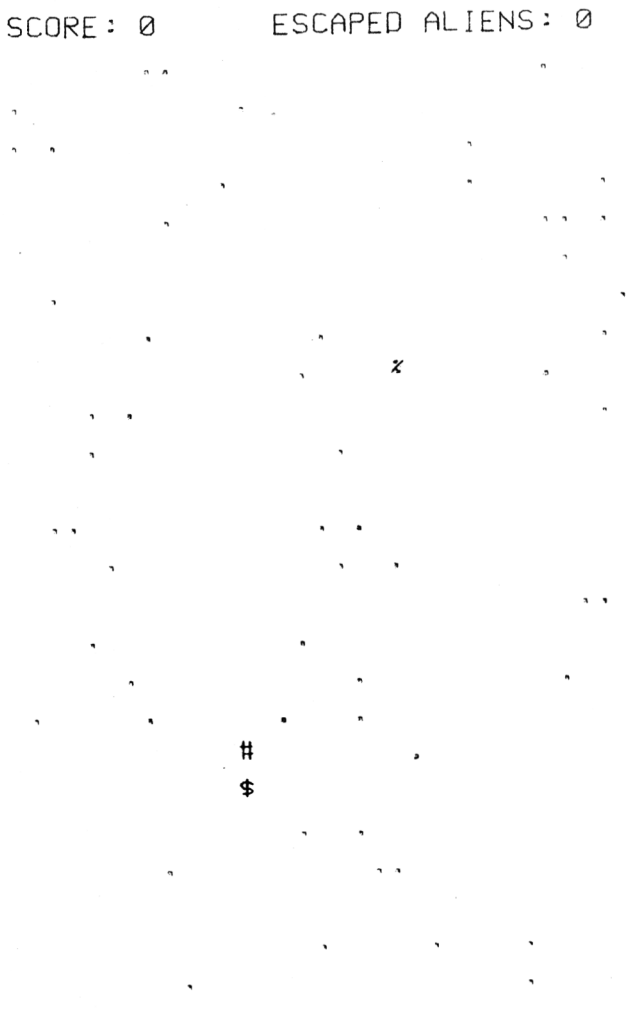
```
10 REM HISSING SID
20 GOSUB 400
30 X$=CHR$(10)+CHR$(11):L=2
40 Y$=CHR$(10)+CHR$(10)
50 PAPER 0:INK 6:CLS
60 MX=INT(RND(1)*38)
70 MY=INT(RND(1)*26)
80 IF SCRN(MX,MY)<>32 THEN 60
90 PLOT MX,MY,"*"
100 PLAY 1,1,2,20
110 FOR I=1 TO 30-L
120 OX=ASC(X$)
130 IF DEEK(783)=48255 THEN OX=OX+1
131 IF DEEK(783)=48351 THEN OX=OX-1
140 OY=ASC(Y$)
150 IF DEEK(783)=48375 THEN OY=OY-1
151 IF DEEK(783)=48319 THEN OY=OY+1
160 IF OX=ASC(X$) AND OY=ASC(Y$) THEN OX
=OX+OX-ASC(MID$(X$,2,1))
161 IF OX=ASC(X$) AND OY=ASC(Y$) THEN OY
=OY+OY-ASC(MID$(Y$,2,1))
165 IF OX<0 OR OX>38 OR OY<0 OROY>26 THE
N 240
170 IF SCRN(OX,OY)=42 THEN L=L+1
180 IF SCRN(OX,OY)=43 HEN 260
182 PLOT OX,OY,"#"
185 PLOT ASC(MID$(X$,L,1)),ASC(MID$(Y$,L
,1)),"  "
190 X$=CHR$(OX)+LEFT$(X$,L)
```

```
200 Y$=CHR$(OY)+LEFT$(Y$,L)
210 NEXT I
220 IF SCRN(MX,MY)=42 THEN PLOT MX,MY,"+
"
230 GOTO 60
240 PLOT 1,0,"YOU CRASHED INTO THE SURRO
UNDING WALL-"
250 GOTO 270
260 PLOT 1,0,"YOU HIT SOMETHING YOU SHOU
LD NOT HAVE-"
270 PLOT 1,2,"AFTER GROWING TO "+STR$(L)
280 FOR A=7 TO 0STEP -1:PAPER A:WAIT 20:
NEXT A
290 PLOT 7,25,"PRESS ANY KEY TO PLAY AGA
IN"
295 IF KEY$<>"" THEN GOTO 295
300 GET A$
310 GOTO 30
400 FOR A=46080+(8*ASC("#"))TO46080+(8*A
SC("#")+7)
410 READ U
420 POKE A,U
430 NEXT A
440 RETURN
450 DATA 51,51,63,18,18,63,51,51
```

ALIEN ESCAPE

In this program, by Daniel Mills, you have to stop the aliens from getting past your command post. If you let more than 5 aliens escape, they will invade earth killing millions of innocent people. Your ship will destroy all aliens on contact, so you must run into them using the cursor keys to control your movements left and right.

This program uses more UDG's than we have ever encountered so far. It is worth typing it in just to see them.


SCORE: 0
ESCAPED ALIENS: 0
%
#
$

Here's the listing for Alien Escape;

```
10 REM ALIEN ESCAPE
20 PAPER 0:INK 6:CLS:GOSUB 500
25 PRINT CHR$(6);CHR$(17)
30 FOR A=1 TO 70:PLOT RND(1)*38+1,RND(1)
*26+1,".":NEXT A
40 FOR A=1 TO 26:PLOT 0,A,INT(RND(1)*7)+
1:NEXT A
45 PLOT 0,20,3:PLOT 0,19,1
50 X=10:S=0
60 WAIT 100
70 P=0:MP=0
80 S=S+P
90 PLOT 1,0,"SCORE:"+STR$(S*10)
100 P=1
110 N=19
120 Y=19
130 C=0
140 PLOT X,20," ":PLOT X,19," "
145 PLOT 1,0,"SCORE:"+STR$(S*10)
146 PLOT 15,0,"ESCAPED ALIENS:"+STR$(MP)
150 IF DEEK(783)=48351 AND X>1 THEN X=X-
1
160 IF DEEK(783)=48255 AND X<38 THEN X=X
+1
170 PLOT X,20,"$":PLOT X,19,"#"
180 PLOT N,P," "
182 PLAY 1,1,1,P*10
185 PLOT 0,INT(RND(1)*17)+1,INT(RND(1)*7
)+1
190 IF RND(1)>.65 THEN N=N+INT(RND(1)*3-
```

```
1)
200 IF N=1 THEN N=2
210 IF N=38 THEN N=37
220 =P+1
225 IF P>25 THEN MP=MP+1:P=1
226 IF MP>5 THEN GOTO 700
230 PLOT N,P,"x"
240 IF N=X AND (P=20 OR P=19 OR P=18) TH
EN 600
250 GOTO 140
500 FOR A=46080+(8*ASC("#")) TO 46080+(8
*ASC("x")+7)
510 READ US:POKE A,US:NEXT A
520 RETURN
530 DATA 0,0,0,4,14,31,31,31
540 DATA 4,4,21,21,21,31,21,4
550 DATA 51,12,18,63,45,0,0,0
600 S=S+7
610 PLOT N,P," "
620 P=1:N=10
630 GOTO 140
700 EXPLODE:CLS
705 PRINT:PRINT:PRINT"SCORE:"S
706 PRINT:PRINT:PRINT
710 PRINT"You have let two many aliens e
scape !"
720 PRINT
730 PRINT"They are now taking over the w
orld.
740 PRINT
```

```
750 PRINT"-------- PRESS ANY KEY FOR NEX
T MISSON
760 IF KEY$<>"" THEN 760
770 GET A$:RUN
```

FOLLOW ME

In this game, you have to duplicate the colour and tone pattern generated by the computer. There is a catch. At first, you will have only one colour/sound combination to remember. Then the computer will add a new combination to the first one, and play both. The sequence will become longer and longer each time you succeed in remembering the sequence.

You copy the computer's pattern by using the keys numbered 1 to 4, (there is a quick reference chart displayed each turn to aid you). There is a high score feature.

```
10 REM FOLLOW ME
20 REM TIM HARTNELL/DANIEL MILLS
30 H=0:PAPER 0:INK 7:CLS
40 DATA 2,4,5,6:RESTORE
50 CLS
60 A$=""
70 S=0
80 D=100
90 M$="###"
100 PRINTM$;TAB(25)CHR$(27)"CSIMON"
110 PLOT 1,3,"SCORE:"+STR$(S)
115 PLOT 17,3,7
120 PLOT 18,3,"HIGH SCORE:"+STR$(H)
130 PLOT 0,4,2
140 PLOT 1,4,"-------------------------
-----------"
150 WAIT D
```

```
160 A$=A$+CHR$(INT(RND(1)*4)+1)
170 FOR T=1 TO LEN(A$)
180 FOR A=1 TO ASC(MID$(A$,T,1))
190 READ B
200 NEXT A
210 RESTORE
220 PLOT 0,11,B+16
230 PLOT 0,15,B
240 PLOT (ASC(MID$(A$,T,1))),15,ASC(MID$
(A$,T,1))
245 PLAY 1,0,1,0
250 MUSIC 1,5,ASC(MID$(A$,T,1))+1,15
260 WAIT 20
270 PLAY 0,0,0,0
280 PLOT 0,11,16
290 PLOT (ASC(MID$(A$,T,1))),15," "
300 NEXT T
310 RESTORE
320 FOR A=1 TO 4
330 PLOT A*6,15,MID$(STR$(A),1)
340 READ B
350 PLOT A*6,16,B+16
360 NEXT A
370 PLOT A*6,16,16
380 RESTORE
390 FOR T=1 TO LEN(A$)
400 FOR A=1 TO D*3
410 I$=KEY$
420 IF I$>"0" AND I$<"5" THEN 600
430 NEXT A
440 M$=MID$(M$,2)
```

```
450 INK 3
460 PLOT 35,0,"\   /"
470 PLOT 35,1," \/ "
480 PLOT 35,2," /\ "
490 PLOT 35,3,"/   \"
500 MUSIC 1,0,7,15:WAIT 20
510 MUSIC 1,0,1,15:WAIT 50
520 PLAY 0,0,0,0
530 WAIT 0:CLS
540 IF M$<>"" THEN 170
550 GOTO 800
600 IF VAL(I$)<>ASC(MID$(A$,T,1)) THEN 4
40
610 MUSIC 1,2,ASC(MID$(A$,T,1))*2,15
620 WAIT 20
630 PLAY 0,0,0,0
640 IF KEY$<>"" THEN 640
650 NEXT T
660 CLS
670 S=S+1
680 D=D-S/5
690 IF D<0 THEN D=0
700 IF H<S THEN H=S
710 GOTO 100
800 ZAP
810 PLOT 1,5,"SCORE:"+MID$(STR$(S),2)
820 PLOT 1,8,"HIGH SCORE:"+MID$(STR$(H),
2)
830 PLOT 1,17,"PRESS ANY KEY TO PLAY AGA
IN"
840 IF KEY$<>"" THEN 840
```

```
850 GET A$
860 GOTO 40
```

ECOLOGICAL DISASTER

Here's your chance to make decisions of national importance. The program explains the starting scenario.

See if you can get the lake to survive longer than 10 weeks.

```
10 REM ECOLOGICAL DISASTER
20 RAN=RND(1):BES=0
25 PAPER 0:INK 3:CLS
30 PRINT "You are the Prime Minister's a
dvisor"
40 PRINT "on natural resources."
50 PRINT
60 PRINT "Part of your job is to stock a
 new"
70 PRINT "lake in the Midlands with fish
 and"
80 PRINT "eels."
90 PRINT
100 PRINT"All very simple, you may think
,  But,"
110 PRINT"there is a problem.  The fish
and the"
120 PRINT"eels are mortal enemies."
130 PRINT
140 PRINT"You must select starting numbe
rs of"
150 PRINT"fish and eels which you think
will"
```

```
160 PRINT"ensure the longest posssible s
urvival"
170 PRINT"of the lake."
180 PRINT
190 PRINT"   Press RETURN when you are r
eady"
200 GET A$:CLS
210 INPUT "Enter the starting number of
eels      (less than 100)";E
ELS
220 IF EELS>100 OR EELS <1 THEN 210
230 INPUT "Enter the starting number of
fish      (less than 100)";F
ISH
240 FISH=FISH/3:WE=0
250 FOR G=0 TO 2:FOR G1=1 TO 12
260 X=INT(RND(1)*5)+16
270 PLOT11,26,"THE LAKE IS EVOLVING"
280 PLOT32,26,16
290 PLOT10,26,X
300 MUSIC 1,G1/2,G+1,15:MUSIC 1,G,G1,15:
MUSIC 1,G1/2,X-10,15
310 WAIT G+G1
320 NEXT G1,G
330 PLAY 1,0,0,0:MUSIC 1,0,10,15:WAIT 50

340 MUSIC 1,0,1,15:WAIT 75:PLAY 0,0,0,0
350 WE=WE+1:CLS
360 PRINT CHR$(27);"T";CHR$(27);"L
     REPORT TO P.M."
370 PRINT
```

```
380 PRINT
390 PRINT "At the end of week ";WE
400 EELS=EELS+((8*EELS-EELS*FISH/3)*RAN)
410 FISH=FISH+((4*FISH-FISH*EELS)*0.01)
420 PRINT
430 PRINT
440 PRINT"------------------------------
--------"
445 IF FISH<0 THEN FISH=0
446 IF EELS<0 THEN EELS=0
450 F=FISH*10:PRINT"FISH:"INT(F)
460 E=EELS*10:PRINT"EELS:"INT(E)
461 IF KEY$<>"" THEN 461
465 GET A$
470 IF EELS<2 OR FISH<2 THEN CLS: GOTO 4
90
480 GOTO 250
490 IF WE>BE THEN BE=WE
495 FOR A=1 TO 200
500 X=INT(RND(1)*20)+1
510 Y=INT(RND(1)*25)
530 PLOT X,Y,"ECOLOGICAL DISASTER"
531 MUSIC 1,0,(Y/3)+1,15
540 PLAY 1,0,0,0
545 NEXT A
550 PAPER 1
560 EXPLODE:CLS
570 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
580 PRINT"IT'S ALL OVER NOW, YOUR LAKE L
OST ITS"
```

```
590 PRINT"LIFE":PRINT
600 PRINT:PRINT"YOUR LAKE STAYED ALIVE F
OR "WE"WEEKS"
610 PRINT:PRINT"THE BEST SO FAR IS "BE
620 PRINT:PRINT:PRINT"PRESS RETURN TO CO
NTINUE."
630 IF KEY$<>"" THEN 630
640 GET A$:GOTO 25
```

CASUALTY

In this program written by Paul Toland and Daniel Mills, you are on a minefield which is full of casualties. You have to push a wheelchair around the minefield, avoiding the mines and electrified fence, to collect each casualty and bring him or her to the hospital (shown as a '+'). The wheelchair can only carry one casualty at a time.

```
10 REM CASUALTY
20 REM PAUL TOLAND
30 REM DANIEL MILLS
35 PRINT CHR$(6);CHR$(17)
40 GOSUB 400
50 INK 7:PAPER 4:CLS:PRINT:PRINT
60 PRINT"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXX"
70 FOR A=3 TO 23:PLOT 1,A,"X":PLOT 36,A,
"X"
80 NEXT A
90 PLOT 1,24,"XXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXX"
95 FOR I=1 TO 20:PLOT INT(RND(1)*34)+2,I
NT(RND(1)*20)+3,"x"
```

```
96 NEXT I
100 FOR I=1 TO 20
110 X=INT(RND(1)*34)+1
120 Y=INT(RND(1)*16)+7
130 IF SCRN(X,Y)<>32 THEN 110
140 PLOT X,Y,"&"
150 NEXT I
160 CAR=0
170 RET=0
180 X=18
190 Y=4
200 A=0
210 D=0
220 PLOT X,Y," "
230 PLOT 18,3,"+"
240 I=DEEK(783)
250 IF I=48351 THEN A=-1:D=0
260 IF I=48255 THEN A=1:D=0
270 IF I=48319 THEN A=0:D=1
280 IF I=48375 THEN A=0:D=-1
285 MUSIC 1,Y/5,(RET+2)/2,10
290 X=X+A:Y=Y+D
300 CH=SCRN(X,Y)
310 PLOT X,Y,35+CAR
315 PLAY1,0,0,0
320 IF CH=37 THEN 500
330 IF CH=88 THEN 520
340 IF CH=38 AND CAR=1 THEN 540
350 IF CH=38 THEN CAR=1
360 IF CH=43 AND CAR=1 THEN CAR=0:RET=RE
T+1:IF RET=20 THEN 560
```

```
370 WAIT 1
380 GOTO 220
400 READ A$:IF A$="Ƶ" THEN RETURN
410 FOR I=0 TO 7
420 READ N:POKE (46080+8*(ASC(A$))+I),N
430 NEXT I
440 GOTO 400
450 DATA #,48,48,40,56,40,40,55,54
460 DATA $,48,48,43,59,43,43,55,54
470 DATA &,0,0,0,0,49,49,63,0
480 DATA ƶ,0,0,0,0,12,30,63,63
490 DATA Ƶ
500 PLOT X,Y,"ƶ"
505 EXPLODE
510 PLOT 1,1,"  YOU HIT A MINE ▯▯▯▯▯▯"
515 GOTO 600
520 PLOT 1,1," zzzƵƵƵƵAAAAAPPPPPPP▯▯▯▯":
ƵAP
530 GOTO 600
540 PLOT 1,1," THE WHEELCHAIR IS OVER-LO
ADED ▯▯"
550 EXPLODE:GOTO 600
560 PING
570 PLOT 1,1," YOU'VE DONE IT ▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯"
600 PAPER 0:INK 4
610 PLOT 9,4,"YOU SAVED "+MID$(STR$(RET)
,2)+" OUT OF 20 ▯"
615 IF KEY$<>"" THEN 615
620 GET A$
630 GOTO 50
```

BOMB RUN II

In this, the last program in the book, you must blow up the city to clear a runway for a safe landing. Press the space bar and a bomb will be dropped, if you are a good shot it will destroy one of the tower blocks. (Go for the tall buildings first).

An interesting ending has been put on this game, so you have an extra incentive to succeed.

```
10 REM BOMB RUN II
20 REM DANIEL MILLS/PETER SHAW
30 GOSUB 9000
40 GOSUB 8000
50 FOR A=0 TO 25
60 FOR B=1 TO 35
70 PLOT B,A," #$"
80 IF KEY$=" " THEN GOSUB BOMB
90 IF SCRN(B+3,A)<> 32 THEN GOTO CRASH
100 WAIT 3
110 PLAY 0,1,1,10
120 NEXT B
130 PLOT B,A,"  "
140 NEXT A
150 PLOT 3,26,"#$":PLOT 5,26,3
160 FOR A=6 TO 15
170 PLOT A,26," %"
180 WAIT 9
190 PLOT A,26," &"
200 WAIT 9
210 NEXT A:PING
215 PLOT A,26,"*"
220 PLOT 4,4,"MISSION COMPLETE - SAFE LA
NDING"
```

```
230 PLOT 7,7,"PRESS A KEY TO PLAY AGAIN"
240 IF KEY$<>"" THEN 240
250 GET A$
260 GOTO 40
300 REM BOMB
310 B1=B
320 FOR DR=A TO 25
330 PLOT B1,DR,"'"
340 B=B+1:IF B=36 THENPLOT B,A,"  ":B=1:
A=A+1
345 IF A=26 THEN 150
346 PLAY 0,1,1,20
347 WAIT 1
350 PLOT B,A," #$"
360 IF SCRN(B+3,A)<>32 THENGOTO CRASH
370 PLOT B1,DR," "
380 NEXT DR
390 RETURN
1000 REM CRASH
1010 EXPLODE
1020 FOR L=A TO 26
1030 PLOT B,L," #$":PLOT B,L-1,"   "
1040 WAIT L
1050 SHOOT
1060 NEXT L
1070 PLOT4,3,"YOU HAVE CRASHED INTO A "+
MID$(STR$(26-A),2)+"0 FOOT"
1080 PLOT12,5,"TOWER BLOCK !"
1090 EXPLODE
1100 IF KEY$<>"" THEN GOTO 1100
1110 GOTO 230
```

```
8000 PAPER 0:INK 4:CLS
8010 BOMB=300:CRASH=1000
8020 FOR LO=1 TO 35
8030 H=INT(RND(1)*10)+15
8040 PLOT LO,H,"("
8050 FOR P=H+1 TO 25
8060 PLOT LO,P,")"
8070 NEXT P
8080 NEXT LO
8090 RETURN
9000 FOR A=46080+8*(ASC("#")) TO 46080+8
*(ASC("*"))+7
9010 READ N
9020 POKE A,N
9030 NEXT A
9031 PRINT CHR$(6);CHR$(17)
9040 RETURN
9050 DATA 49,56,57,63,63,15,16,0
9060 DATA 62,40,18,58,63,58,34,16
9070 DATA 6,6,4,8,8,12,12,12
9080 DATA 6,6,4,14,8,12,10,18
9090 DATA 0,18,18,12,30,30,30,12
9100 DATA 0,12,30,63,45,63,45,63
9110 DATA 45,63,45,63,45,63,45,63
9120 DATA 14,14,4,31,4,10,17,0
```

CHAPTER SEVENTEEN USING THE SYSTEM

This is the last chapter in the book, and in it we'll discover about things such as DEEK, DOKE, CALL and USR.

As you may have gathered, the computer's memory is like a lot of matchboxes with numbers in. At the very top of this pile of matchboxes (the top 16383 matchboxes to be precise), is what's called 'ROM'. ROM stands for Read Only Memory, that is you can look at it, but you can't alter it. The computer uses this to interpret what you are telling it to do, and to read the keyboard etc.

Below the ROM there are things like the SCREEN, i.e. every character displayed on the screen at any one time is stored in its own location of memory. Each character has its own little match box. These areas of memory can be changed, so the matchbox is being continually opened, filled, closed, opened, emptied, etc, etc. This area of memory is called RAM, Random Access Memory. Which means you can open any matchbox you like. You don't need to unpile the lot to get to any one box. You can, as it were, open them from the side.

PEEK

To look at what's inside any matchbox from Ø–65535, you use PEEK (m), where 'm' is the matchbox you are talking about.

POKE

To put something inside a matchbox use POKE m,n. 'm' is the location you want to put your number, and n is your number, (this must be between Ø–255).

DEEK

As Ø–255 is not a very large number range, DEEK is used to give a Double pEEK. DEEK(n) gives the contents of the n matchbox plus 256 times the contents of the next one. i.e. DEEK (n) gives;

PEEK(n) + 256*PEEK(n + 1)

DOKE

DOKE is a Double POKE. DOKE m,x puts the whole number (x/256) into m + 1 and the remainder goes into m, i.e.;

POKE(m + 1),INT (x/256):POKE m,(x – PEEK(m + 1))

CALL

Once you have written a machine code program (don't worry about machine code just yet), CALL x transfers, or RUN's your machine code program.

DEF USR

Defines the start of the machine code routine.

FRE

PRINT FRE(Ø) returns the amount of matchboxes left empty, i.e. how much memory you have left.

To find out more about 'machine code', read Chapter 13 of the Oric programming manual.

APPENDIX: USING MATHS

Here is a summary of the mathematical symbols on your computer.

USUAL SYMBOL	COMPUTER SYMBOL
+ (plus)	+
− (minus)	−
× (multiply)	*
÷ (divide)	/
M^n (raise to power)	M↑n

GETTING STARTED ON YOUR ORIC
is one of a series designed by experts and put together by actual users which will allow the first-time buyers of a personal computer to make effective, creative and constructive use of their new acquisition with the minimum of time-wasting false starts and the maximum of personal satisfaction and fulfilment. Free of jargon, this is a simple to read, easy to use, step by step guide to proficiency in the use of the ORIC which will enable readers to obtain the maximum of results from their machine in the least possible time.

Also in this series

GETTING STARTED ON YOUR ZX81
GETTING STARTED ON YOUR SPECTRUM
GETTING STARTED ON YOUR BBC MICRO
GETTING STARTED ON YOUR TRS 80/DRAGON
GETTING STARTED ON YOUR COMMODORE/VIC 20
GETTING STARTED ON YOUR ATARI

Futura Publications
Non-fiction
0 7088 24471

GB £ NET +002.95

ISBN 0-7088-2447-1

00295

9 780708 824474

GETTING STARTED ON YOUR ORIC Tim Hartnell & Peter Shaw